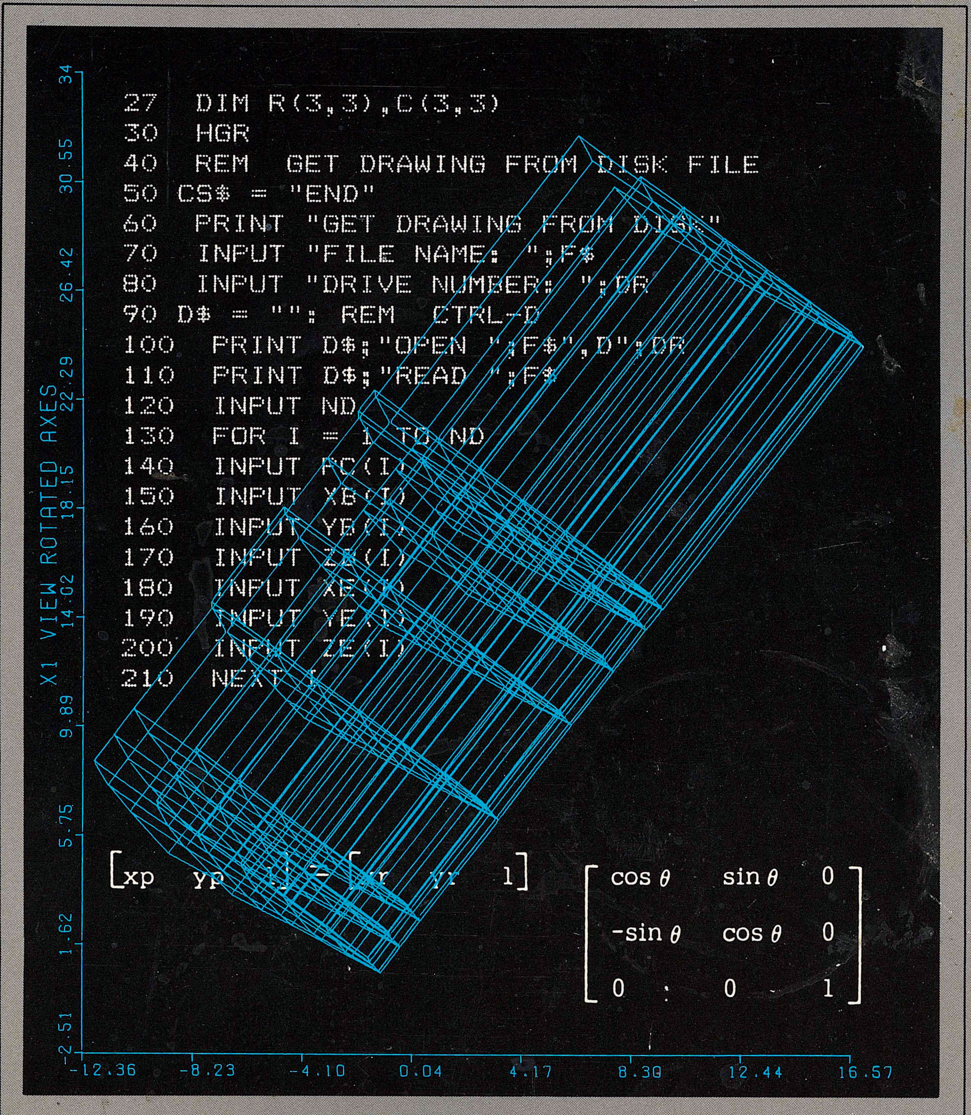
ENGINEERING SOFTWARE FOR MICROS



```
27  DIM R(3,3),C(3,3)
30  HGR
40  REM  GET DRAWING FROM DISK FILE
50 CS$ = "END"
60  PRINT "GET DRAWING FROM DISK"
70  INPUT "FILE NAME: ";F$
80  INPUT "DRIVE NUMBER: ";DR
90 D$ = "": REM  CTRL-D
100  PRINT D$;"OPEN ";F$",D";DR
110  PRINT D$;"READ ";F$
120  INPUT ND
130  FOR I = 1 TO ND
140  INPUT RC(I)
150  INPUT XB(I)
160  INPUT YB(I)
170  INPUT ZB(I)
180  INPUT XE(I)
190  INPUT YE(I)
200  INPUT ZE(I)
210  NEXT I
```

$$\begin{bmatrix} xp & yp & 1 \end{bmatrix} = \begin{bmatrix} xr & yr & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

X1 VIEW ROTATED AXES
34
30.55
26.42
22.29
18.15
14.02
9.89
5.75
1.62
-2.51
-12.36
-8.23
-4.10
0.04
4.17
8.30
12.44
16.57

B. J. Korites

construction systems technology
DEPARTMENT

ENGINEERING SOFTWARE FOR MICROS

by

B. J. Korites

KERN PUBLICATIONS

190 Duck Hill Road
Duxbury, MA 02332

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

The authors have taken due care in preparing this book and the programs in it including research and testing to insure their accuracy and effectiveness. The author and the publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation of them in this book. Further, neither the author nor the publisher make any warranty of any kind, expressed or implied, with regard to the accuracy of the programs in this book or on the computer diskette which accompanies this book. In no event shall the author or the publisher be liable for incidental or consequential damages in connection with or arising from the furnishing, performance, or use of the programs in this book or on the accompanying diskette.

COPYRIGHT NOTICE

Copyright © 1982 by Kern Publications. No part of this publication, or the accompanying computer diskette, may be reproduced, stored in an information retrieval system, or transmitted in any form or by any means electronic, mechanical, photocopying, recording or otherwise without the express written permision from the publisher. However, permission is granted by the publisher to original purchasers of this publication and the accompanying computer diskette to make backup copies of this publication and the accompanying computer diskette for their own personal use. This permission is not to be construed as a license to copy and distribute these programs in any form to third parties and such activity is expressly prohibited.

TRADEMARK NOTICE

Apple is a trademark of the Apple Computer company.

CONTENTS

Chapter 1: Fundamentals of Computer Graphics

Introduction, *1-2*
Coordinate Systems, *1-2*
Graphics Modes, *1-4*
Plotting Commands, *1-4*

PLOT10:	Plots a point at the center of the screen, *1-5*
PLOT11:	Plots points at the corners of the Page 1 hi-res screen, *1-6*
PLOT12:	Defines the plotting screen by drawing lines between the corners, *1-6*
PLOT13:	Illustrates use of the chain line command, *1-7*
PLOT14:	Draws screen border and labels x and y axes, *1-7*
PLOT15:	Draws a line of points in the x direction, *1-8*
PLOT16:	Draws a line of points in the y direction, *1-9*
PLOT17:	Draws a line of points between two arbitrary points, *1-12*
PLOT60:	Draws a circle using points, *1-14*
PLOT61:	Draws a circle using lines, *1-16*
PLOT62:	Draws a polygon using lines, *1-18*

Chapter 2: CAD/CAM Software

CAD/CAM Systems, *2-2*
CAD/CAM on Micros, *2-2*
Limitations of Micros, *2-3*
CAD/CAM System Organization, *2-4*
The Drawing Data Base, *2-5*
Real vs Binary Data, *2-5*
Applications Programs, *2-5*
Drawing Revisions, *2-9*
Advantages of CAD/CAM Systems, *2-10*

CAD10:	Interactively creates a 3-dimensional drawing in Apple coordinates and stores on disk, *2-11*
CAD20:	Interactively creates a 3-dimensional drawing in real physical coordinates and stores on disk, *2-19*
CAD30:	Recalls a 3-dimensional drawing from disk, updates it and replaces it on disk, *2-30*
CAD40:	Recalls several 3-dimensional drawings from disk, displays them simultaneously, and positions them on the screen, *2-43*
CAD50:	Recalls a 3-dimensional drawing from disk, prints the drawing data and adds physical properties to members, *2-48*
CENTROID:	Calculates total weight and the centroid location of a 3-dimensional structure and plots the centroid location, *2-58*
ISOVIEW:	Recalls a 3-dimensional drawing from disk and rotates it in 3 dimensions, *2-65*

Chapter 3: Mechanisms

Introduction, *3-1*
- **LINKAGE1:** Static simulation of the motion of a four bar linkage, *3-3*
- **LINKAGE2:** Static simulation of the motion of a four bar linkage with a follower bar, *3-9*
- **LINKAGE3:** Dynamic simulation of the motion of a four bar linkage, *3-16*
- **LINKAGE4:** Dynamic simulation of the motion of a slider crank, *3-19*

Chapter 4: Fourier Analysis

- **FOURIER1:** Evaluates the first five terms of the Fourier Series for a periodic waveform and plots the magnitudes, *4-1*
- **FOURIER2:** Evaluates the continuous frequency spectrum of the Fourier Transform of a non-periodic waveform and plots the spectrum, *4-8*

Chapter 5: Matrix Methods

Gauss-Jordan Elimination, *5-1*
- **MATRIX1:** Solves a set of simultaneous linear equations using the Gauss-Jordan Elimination technique, *5-9*

Thermal Analysis, *5-15*

Structural Analysis, *5-20*
- **MATRIX2:** Recalls a matrix from disk, updates it and replaces it on disk, *5-24*

Chapter 6: Optimization

Introduction, *6-1*
- **OPTIMIZER1:** Finds and plots the shortest path between a number of points using the "brute force" approach, *6-1*
- **OPTIMIZER2:** Finds and plots the shortest path between a number of points using the Monte Carlo method, *6-10*

FOREWORD

This is a collection of programs for engineering design and analysis on microcomputers. It has been written for practicing engineers and scientists, professional software developers, educators and students.

Actually, this is more than just a collection of programs: It is a comprehensive approach to the use of micros for engineering applications. It shows how to circumvent the weaknesses of micros while exploiting their strengths to the fullest.

Most engineers are accustomed to working with large computer systems where little attention is payed to memory size or processing speed. With micros, however, things are different: Their active memories are relatively small, ranging anywhere from 48K up to 350K or so, and their processing speeds are relatively slow. But micros have some advantages which, for many applications, greatly offset these limitations: They are inexpensive, they do superb graphics, and if you have one on your desk, the turn around time is practically instantaneous. While I don't want to imply that micros can be used effectively in *all* engineering environments, they have some unique characteristics which make them extremely valuable tools in a great many situations.

When putting together this collection of programs I had three primary objectives in mind: First, I wanted to show how a micro's limited memory can be worked around by clever use of disk storage. Second, I wanted to show what a tremendous advantag the micro's graphics capability can be when used to interactively create engineering data files and graphically display the results of analyses. Third, I wanted to present a few applications programs, such as Gauss-Jordan elimination and Fourier analysis, which you can use as-is for the solution of real-life engineering problems.

Many of the concepts and techniques presented here have been extrapolated directly from large system CAD/CAM software technology. The basic idea is to create a data file of information, store it on disk, and operate on it with various applications programs. The data file, which we will call an "engineering drawing", is a complete geometric and physical description of a 2 or 3 dimensional system such as an electric circuit or a building structure. The idea of interactively creating this drawing, storing it on disk, and then operating on it with application programs is the basic concept that I wish to develop with this collection of programs.

In Chapter 1, we review the fundamentals of computer graphics. In Chapter 2, a series of programs are developed which can be us-

More Software Available From Kern Publications

★ **Graphic Software For Microcomputers** - This self-teaching guide will show you how to write your own 2 and 3 dimensional graphics software. It contains 61 fully documented programs in Applesoft BASIC to interactively create 2 and 3 dimensional shapes, translate, rotate, scale, stretch, clip, remove hidden lines. Also, perspective views, shading, surface intersections, tablet software, animation, applications to business, science engineering, and much more. A complete graphics tool kit for professionals, students and software developers. Adopted as a text by numerous schools and universities including Stanford, U. North Carolina and Texas A+M. Written by B.J. Korites,PhD.

"One of the most outstanding books on computer software"
- Alan Grund, University of Illinois -

"The best book available on microcomputer graphics"
- Creative Computing, Feb, 1982 -

Book: $21.95
Disk: $19.95 (Specify Apple II+ DOS3.3 or IBMpc DOS1.1)

★ **Data Plotting Software** - A collection of programs to process and display all types of data: bar charts, stock market charts, engineering and scientific data, 3D views of surfaces, pie charts, sorting, filtering, running averages, curve fitting and more. Applications to business, science and engineering. All programs in Applesoft BASIC fully documented and keyed to theory. Use as is or modify for your own applications.

Book: $24.95
Disk: $19.95 (Specify Apple II+ DOS3.3 or IBMpc DOS1.1)

★ **Structural Analysis Software** - This self-teaching guide contains a series of fully-documented programs which calculate stresses and deflections in beams, plates, trusses and frames using classical and matrix techniques. All programs in Applesoft BASIC fully documented and keyed to theory and equations. Use them as is or modify for your own applications.

Book: $39.95
Disk: $19.95 (Specify Apple II+ DOS3.3 or IBMpc DOS1.1)

To Order

To order send check drawn on US bank, money order in US funds, **VISA** or **MASTERCARD** number with expiration date to **KERN PUBLICATIONS**, 190 Duck Hill Road, P.O. Box 1029, Suite EA, Duxbury, MA 02332. Please add $2 per book for 4th class postage in US and Canada; $3 1st cl or UPS in US; $4.50 1st cl Canada; $12 air Europe and Central America; $18 elsewhere. When ordering diskettes, please be sure to specify Apple or IBMpc. For faster delivery, call (617)934-0445.

COMPUTER EQUIPMENT

The computer equipment used in developing this software consisted of an Apple II+ microcomputer with 48K of memory, a NEC monitor, one disk drive, and an Epson MX-80 dot matrix printer. All programs have been written in straightforward BASIC with a minimum of machine-dependent commands so, with minor modification, they should be compatible with any computer that has graphics capability. The modifications you will have to make will be primarily in the graphics commands and the commands that control peripherals such as the printer and plotter.

All the computer drawings are simple screen dumps and were produced on the Epson MX-80 with a screen dump program produced by Computer Stations Inc. In order to obtain hard copy graphics on your printer you will need a similar screen dump program.

You may notice that some of the drawing may not appear to have the proper proportions. For example, a circle looks more like an elipse. This is because of the characteristics of the NEC monitor's screen combined with the Apple's plotting characteristics. A circle having a diameter of 10 plotting units should appear 10 units across by 10 units high. But on the NEC monitor, the circle appears to be about 8 units high.

The problem is, the horizontal spacing of the plot points on the monitor is not the same as the vertical spacing. You might imaging the screen as been a finely drawn checkerboard. When we plot a point at a certain location, we turn on one of the blocks. When we draw a line, we turn on a series of blocks. In the case of the Apple, there are 280 blocks horizontally and 160 vertically in the hi-res mode. Unfortunately, with the NEC monitor, the blocks are not square. As a result, when we try to draw a square, it appears as a rectangle; a circle as an elipse.

Different monitors combined with different computers behave differently in this regard. When the IBMpc is used with the NEC monitor the distortion problem is worse because the IBM, in its hi-res mode, tries to squeeze 640 plotting units horizontally across the screen while the Apple uses 280.

Most people don't bother to correct the problem as long as it isn't too severe. But if you do want to eliminate the distortion, the way to do it is to scale the x or y values before plotting. For example, to draw a circle of points, we would calculate the proper coordinates for the points, each of which would be the same mathematical distance from the circle's center, and use the HPLOT x,y command to place the points at the proper x,y coordinates on the Apple's screen. As we just saw, the result will be an elipse. To eliminate the distortion multiply either x or y by a scale factor to stretch or compress the drawing appropriately. To stretch in the y direction by 20 percent, we would change the HPLOT x,y command to HPLOT x,y*1.2. To contract it in the y direction by 20 percent we would use HPLOT x,y*.8.

The easiest way to obtain the proper scale factor is to run program *PLOT60,* which draws a circle, and try different scale factors in line 210 of that program until a true circle is obtained. Once the correct scale factor has been determined for your system, introduce it in all HPLOT commands in the other programs.

IMPORTANT NOTICE !

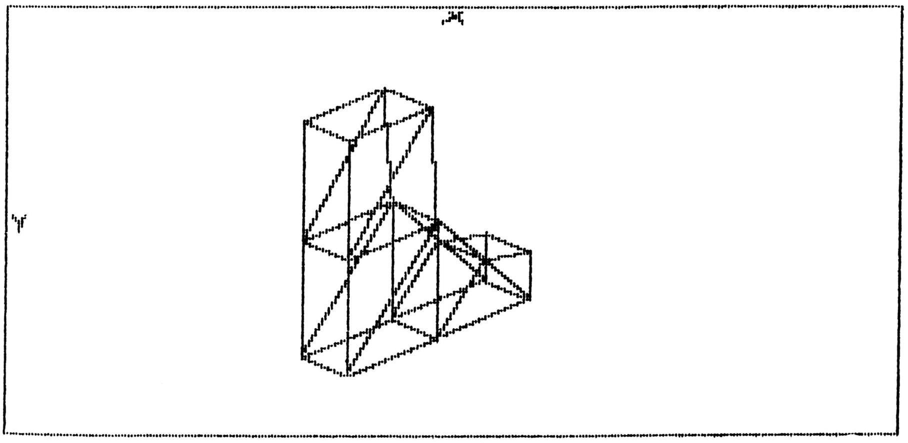
PROGRAM DISKETTE AVAILABLE

This book contains several program listings. A diskette is available which will save you the trouble of typing them into your computer. The diskette is available in 5¼ inch size for the Apple II+ DOS3.3 and the IBMpc DOS1.1. Suitable modifications have been made to the Applesoft listings in this book to enable the diskette to run on the IBMpc.

To order send $19.95 in the form of a check drawn on US bank, money order in US funds, **VISA** or **MASTERCARD** number with expiration date to Kern Publications, 190 Duck Hill Road, P.O. Box 1029, Suite EA, Duxbury, MA 02332. Please be sure to specify Apple or IBMpc. For faster delivery call (617)934-0445.

Chapter 1: Fundamentals of Computer Graphics

Introduction, *1-2*
Coordinate Systems, *1-2*
Graphics Modes, *1-4*
Plotting Commands, *1-4*
- **PLOT10:** Plots a point at the center of the screen, *1-5*
- **PLOT11:** Plots points at the corners of the Page 1 hi-res screen, *1-6*
- **PLOT12:** Defines the plotting screen by drawing lines between the corners, *1-6*
- **PLOT13:** Illustrates use of the chain line command, *1-7*
- **PLOT14:** Draws screen border and labels x and y axes, *1-7*
- **PLOT15:** Draws a line of points in the x direction, *1-8*
- **PLOT16:** Draws a line of points in the y direction, *1-9*
- **PLOT17:** Draws a line of points between two arbitrary points, *1-12*
- **PLOT60:** Draws a circle using points, *1-14*
- **PLOT61:** Draws a circle using lines, *1-16*
- **PLOT62:** Draws a polygon using lines, *1-18*



INTRODUCTION

The emphasis throughout this book will be on graphical display of information. We will be developing software to graphically create information on the computer, process the data, and graphically display results. Consequently, it is essential that you have a strong understanding of elementary computer graphics techniques and a familiarity with plotting commands.

In this chapter we will be developing a series of programs that illustrate the use of the various plotting commands that are available on the Apple II computer. If you have a different type of computer, you will most likely find that the plotting commands are similar in function although they may be different in syntax. We will also be introducing some important concepts which are commonly used in computer graphics operations. Most notable are vectors. We will be covering vectors, unit vectors and various vector operations such as addition, dot products and cross products. These will be used throughout the remainder of this book.

COORDINATE SYSTEMS

Before turning to the programs, we should first discuss the Apple's plotting screen and its coordinate system. The plotting screen is, of course, two dimensional. A point on the screen is defined by two coordinates which we will refer to as x and y. Most people, especially engineers, are accustomed to seeing the coordinate system's origin located at the lower left corner of the plotting area with the x axis extending from left to right along the bottom and the y axis extending from bottom to top along the left side. However, as can be seen in Figure 1.1, the Apple's coordinate system has its origin at the upper left corner. The x axis goes along the top of the screen from left to right and the y axis goes down the left side.

Since we will be working with 3 dimensional shapes, we must introduce a third coordinate, z. As seen in Figure 1.1, the z axis goes into the screen. This follows the right hand rule convention: The direction of z is obtained by rotating the x axis into the y axis through the smaller angle and following the direction of motion of a right handed screw.

A 3 dimensional object is defined mathematically by data points in x,y,z space. The computer's plotting screen knows nothing about the z axis. We obtain a 2 dimensional view of a 3 dimensional object by projecting the x,y coordinates of the data points onto the plotting screen. That is, we simply plot the x and y coordinates of the data points and connect the appropriate points by lines.

Later, we will be working with 3 dimensional graphics. The Z axis will go into the screen as shown. This follows the right-hand rule convention. That is, the direction of Z is obtained by rotating X into Y through the smaller angle and progressing in the same direction that would be followed by a right handed screw.

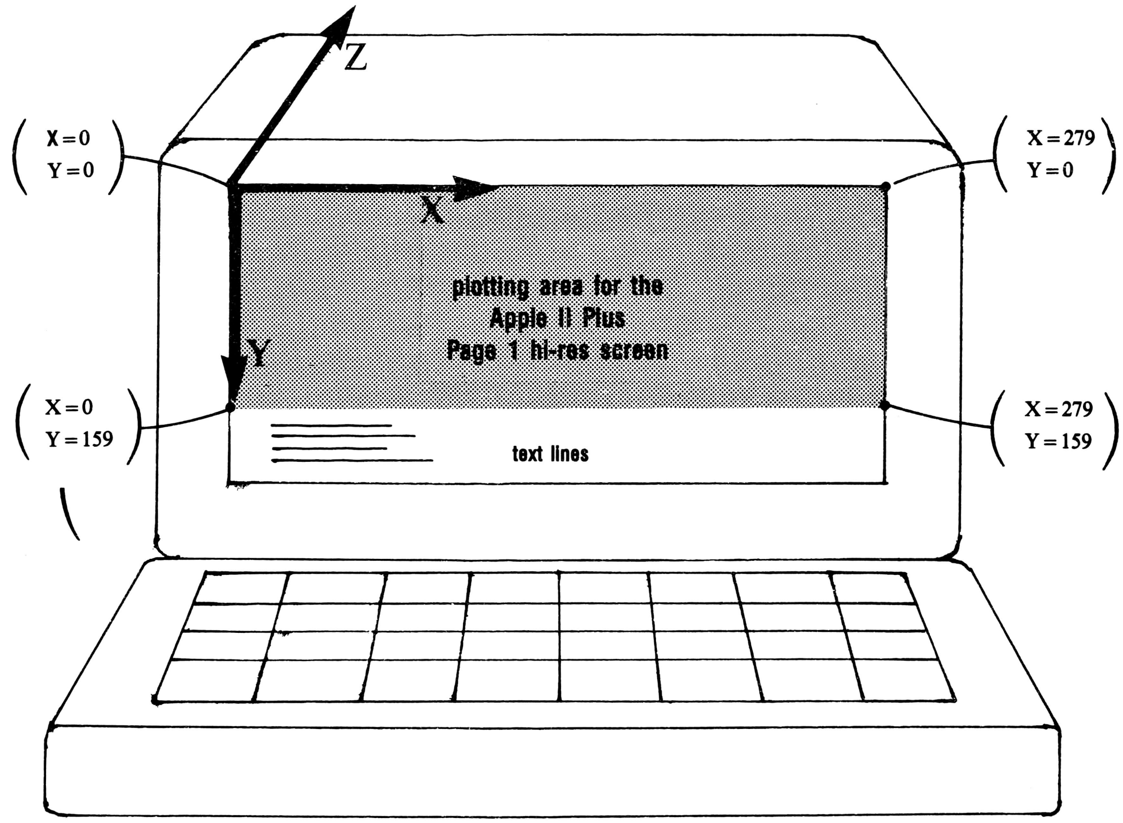

Z
X=0
Y=0
X=279
Y=0
X
plotting area for the
Apple II Plus
Page 1 hi-res screen
Y
X=0
Y=159
X=279
Y=159
text lines

Figure 1-1: PLOTTING SCREEN FOR THE APPLE II PLUS

GRAPHICS MODES

Three graphics modes are possible on the Apple II computer. The first is a low resolution mode where pictures are composed of crude blocks. It is a popular mode in games software since there isn't much information on the screen and animation programs can be made to run very quickly. However, because of the coarseness of the drawings, this mode isn't much use it engineering or other technical work. We won't be using the low-res mode in this book.

The second mode is page 1 hi-resolution graphics. In this mode plotting values of x may range from 0 to 279; y may range from 0 to 159. We may place a point or draw a line anywhere on the screen within these limits. If either x or y lies outside these limits an error message will result and program execution will stop. An exception to this is that y may extend to 191 without an error being indicated. In page 1 hi-res graphics four lines of text are displayed at the bottom of the screen. These are useful for messages indicating what the program is doing and for prompts indicating what input data or commands are required by the program.

The third mode available is page 2 hi-resolution graphics. In this mode, legal values of x range from 0 to 279 (the same as page 1) but y may extend from 0 to 191. In this mode the four text lines at the bottom of the page are obliterated, a serious disadvantage. Another shortcoming of page 2 graphics is that it requires more memory. This limits the size of the program that can be loaded and the amount of program data that can be stored. In view of these two problems, page 2 graphics will not be used in this book. We will use only the page 1 hi-res mode.

PLOTTING COMMANDS

We can do just about anything we want graphically with the following five commands:

> **HGRn** : This command sets the graphics mode. n may be blank or 2 corresponding to page 1 or page 2 hi-res graphics. As previously mentioned, we will be using only page 1 hi-res graphics.
>
> **HCOLOR = n** : This sets the plotting color. Each computer has its own code for colors and color graphics can be obtained by inserting the proper value for n. For example, n = 2 produces blue on the Apple. This command can be inserted anywhere in a program to change the plotting color. We will be

using the values 0, which produces black on a black background (to erase plots) and 7, which produces white on a black background. Keep in mind, however, that you can change these values to produce color if you have a color monitor.

HPLOT x,y : This commands plots a point at screen coordinates x,y.

HPLOT x1,y1 TO x2,y2 : This draws a line from x1,y1 to x2,y2.

HPLOT x1,y1 TO x2,y2 TO ....xn,yn : This draws a chain of lines from x1,y1 to x2,y2 ...etc.

The programs which follow illustrate the use of these plotting commands. If you already have a working familiarity with plotting, you may skim over most of these programs. However, your attention is called to programs PLOT17 and PLOT60. The first of these is a program that draws a line of points between any two arbitrary points on the screen. We use vectors to do this. You should review that program since it introduces the concept of a vector and the terminology that will be used throughout much of this book. The second of these programs, PLOT60, draws a circle using points. It illustrates the use of trig functions in plotting and the use of relative cylindrical coordinates.

PROGRAM *PLOT10*

This is a simple program that plots a point at the center of the screen. It illustrates use of the HGR, HCOLOR and HPLOT functions.

Line 50 contains the HGR commands which puts the computer into the graphics mode. Note that when this program line is executed the screen is cleared of text, except for the bottom four lines. To exit from the graphics mode, type TEXT.

Line 60 contains the HCOLOR command which sets the plotting color to white.

Line 70 contains the HPLOT x,y command which plots the point at the screen coordinates x = 140, y = 80, which is the center of the Apple's screen.

PLOT10
PLOT11
PLOT12

```
   10  PRINT "PLOT10"
   20  PRINT "PLOTS A POINT"
   30  PRINT "AT THE CENTER"
   40  PRINT "OF THE PLOTTING AREA"
   41  PRINT "TYPE CONT TO CONTINUE"
   42  STOP
➤  50  HGR
➤  60  HCOLOR= 7
➤  70  HPLOT 140,80
   80  END
```

PROGRAM *PLOT11*

This program shows the boundaries of the plotting area by placing points at the four corners (0,0), (279,0), (279,159), (0,159).

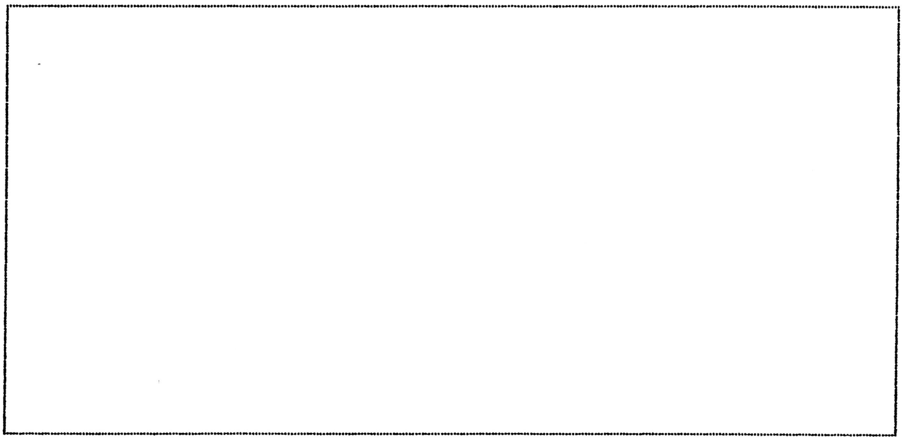
```
   10  PRINT "PLOT11"
   20  PRINT "PLOTS POINTS AT"
   30  PRINT "THE CORNERS OF THE"
   40  PRINT "PLOTTING AREA"
   50  PRINT "TYPE CONT TO CONTINUE"
   60  STOP
   70  HGR
➤  80  HPLOT 0,0
➤  90  HPLOT 279,0
➤  100  HPLOT 279,159
➤  110  HPLOT 0,159
   120  END
```

PROGRAM *PLOT12*

This program illustrates use of the HPLOT x1,y1 TO x2,y2 command by drawing lines between the four corners of the plotting area.

```
   10  PRINT "PLOT12"
   20  PRINT "DRAWS LINES BETWEEN"
   30  PRINT "THE FOUR CORNERS OF"
   40  PRINT "THE PLOTTING AREA"
   50  PRINT "TYPE CONT TO CONTINUE"
   60  STOP
   70  HGR
   80  HCOLOR= 7
➤  90  HPLOT 0,0 TO 279,0
➤ 100  HPLOT 279,0 TO 279,159
➤ 110  HPLOT 279,159 TO 0,159
➤ 120  HPLOT 0,159 TO 0,0
  130  END
```

PLOT12
PLOT13
PLOT14



PROGRAM *PLOT13*

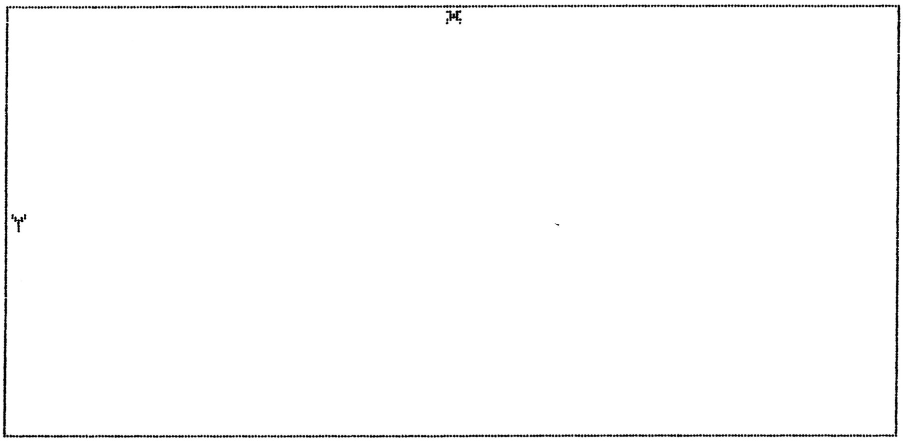
This program illustrates use of the HPLOT x1,y1 TO x2,y2 TO....xn,yn command by drawing a chain of lines between the four corners of the plotting area. It accomplishes the same result as *PLOT12* but in one line of coding instead of four.

```
  10  PRINT "PLOT13"
  20  PRINT "DRAWS LINES BETWEEN THE"
  30  PRINT "FOUR CORNERS OF THE"
  40  PRINT "PLOTTING AREA USING THE"
  50  PRINT "CHAIN PLOTTING COMMAND"
  60  PRINT "TYPE CONT TO CONTINUE"
  70  STOP
  80  HGR
  90  HCOLOR= 7
➤ 100  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
  110  END
```

PROGRAM *PLOT14*

This program is similar to *PLOT13* except that it labels the x and y axes. This illustrates how alphanumeric characters can be generated by simple plotting commands.

```
10  PRINT "PLOT14"
20  PRINT "DRAWS A FRAME AROUND THE"
30  PRINT "PLOTTING AREA AND LABELS"
40  PRINT "THE AXES"
50  PRINT "TYPE CONT TO CONTINUE"
60  STOP
70  HGR
80  HCOLOR= 7
90  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
➤ 100  REM  PLOT "X" AT CENTER OF X AXIS
110  HPLOT 138,2 TO 142,6
120  HPLOT 142,2 TO 138,6
➤ 130  REM  PLOT "Y" AT CENTER OF Y AXIS
140  HPLOT 2,77 TO 4,80 TO 4,83 TO 4,80 TO 6,77
150  END
```

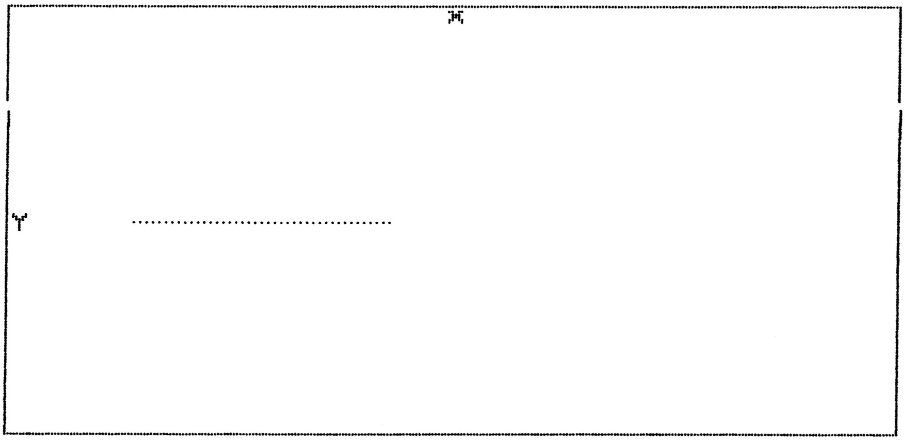


PROGRAM *PLOT15*

This program illustrates how a line can be drawn using points. The line extends in the x direction. Line 95 sets the y coordinate of the points to 80. Loop 100-120 increments x from 40 to 120 in steps of 2. In other words, the points comprising the line will run from x = 40 to x = 120 and will be 2 plotting units apart. Line 110 does the plotting using the familiar HPLOT x,y command.

```
10  PRINT "PLOT15"
20  PRINT "DRAWS A LINE OF POINTS"
30  PRINT "IN THE X DIRECTION"
40  PRINT "TYPE CONT TO CONTINUE"
50  STOP
60  HGR
70  HCOLOR= 7
80  GOSUB 9990
95 Y = 80
```

```
100  FOR X = 40 TO 120 STEP 2
110  HPLOT X,Y
120  NEXT X
130  GOTO 9998
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 4,80 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```



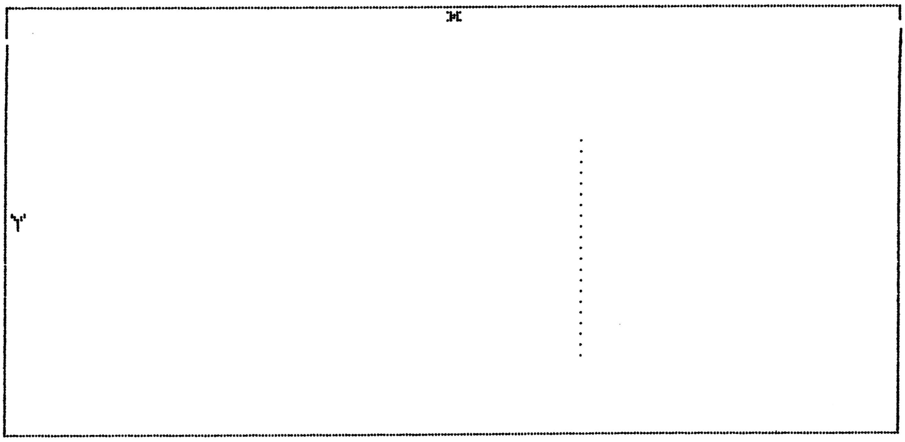
PROGRAM *PLOT16*

This program is similar to *PLOT15* except that the line goes in the y direction. Line 95 sets the value of x and loop 100-120 increments values of y. Note that the spacing between the dots has been increased to 4 units.

```
10  PRINT "PLOT16"
20  PRINT "DRAWS A LINE OF POINTS"
30  PRINT "IN THE Y DIRECTION"
40  PRINT "TYPE CONT TO CONTINUE"
50  STOP
60  HGR
70  HCOLOR= 7
80  GOSUB 9990
95 X = 180
100  FOR Y = 50 TO 130 STEP 4
110  HPLOT X,Y
120  NEXT Y
130  GOTO 9998
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
```

PLOT16
PLOT17

```
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 4,80 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```



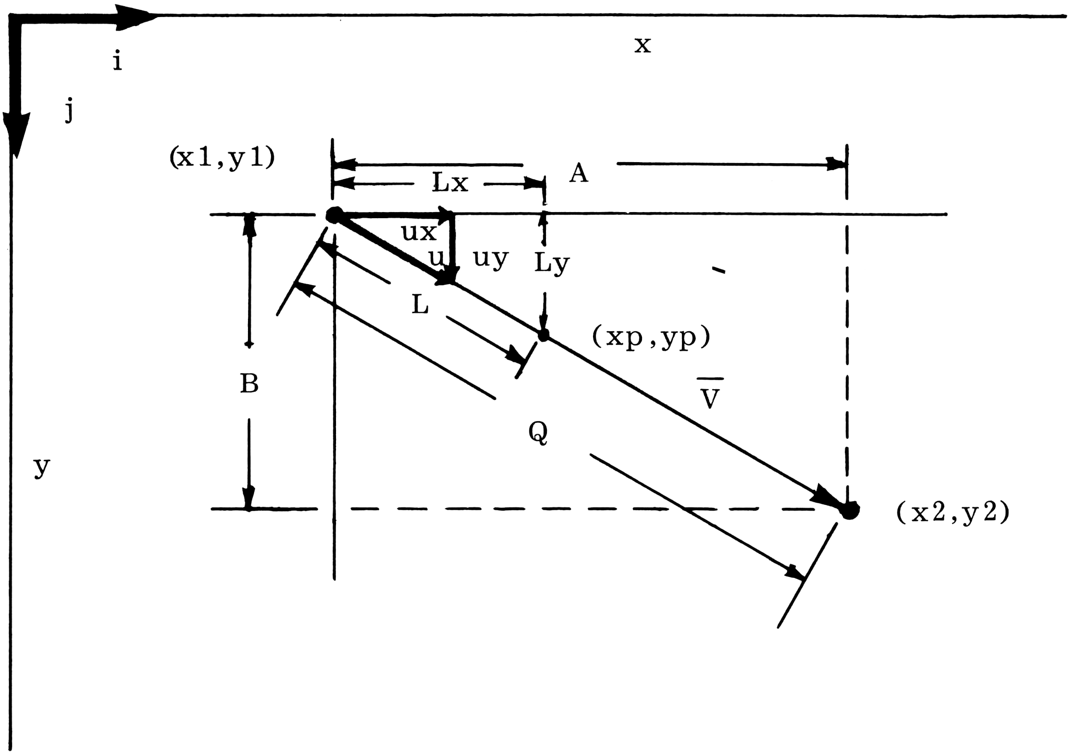
PROGRAM *PLOT17*

This program draws a line of points between two arbitrary beginning and end points which may be specified by the user. This program is important since it introduces the concept of vectors, unit vectors, and the associated terminology and programming style we will be using throughout this book.

In program *PLOT15,* when we plotted aline of points in the x direction it was easy to calculate the coordinates of each point. We simply set the value of y in the program and let a loop calculate values of x for us. Similarly for program *PLOT16* where we plotted a line of points in the y direction. But in this present program, since we are plotting an inclined line of points that may have arbitrary orientation, we must develop an algorithm that will automatically calculate the coordinates of the points being plotted for arbitrary beginning and end points. We will accomplish this using vectors.

The geometry is shown in the following sketch. x1,y1 are the coordinates of the beginning of the line; x2,y2 of the end. Given these coordinates, we will write a program that will calculate the length of the line, Q; the x projection, A; and the y projection, B. From these quantities we will get ux and uy, which are the x and y components of a unit vector directed along the line, u. Recall that a unit vector has a magnitude, or length, equal to 1. Since ux and uy are the projections of this unit vector on the x and y axes, they are equal to the cosines of the angles between u and the x and y axes respectively. For this reason, ux and uy are often called "direction cosines".

Once we have obtained these quantities, it is a simple matter to calculate the coordinates of a point on the line. Suppose the point is located a distance L from the beginning of the line. Its global position is the sum of two vectors: One running from 0,0 to x1,y1; the other from x1,y1 to xp,yp. Its x coordinate, xp, is therefore the sum of the x components of these two vectors. Similarly for the y coordinate, yp:



$$\overline{V} = Ai + Bj$$

$$A = x2 - x1$$

$$B = y2 - y1$$

$$Q = SQR(A*A + B*B)$$

PLOT17

$$u = \overline{V}/Q$$

$$u = uxi + uyj$$

$$xp = x1 + Lx$$
$$yp = y1 + Ly$$

From similar triangles:

$$Lx = ux*L$$
$$Ly = uy*L$$

$$xp = x1 + ux*L$$
$$yp = y1 + uy*L$$

In the program, lines 100, 110 and 120 form the input stage where the beginning and end coordinates are specified as well as the spacing between the points, D.

Lines 130 and 140 calculate the x and y components of the vector from point 1 to point 2. Line 150 calculates Q, the length of the line, which is the same as the length of the vector from 1 to 2. This is done by taking the square root of the sum of the squares.

Lines 160 and 170 calculate the components of the unit vector, u, which is oriented in the direction of the vector from 1 to 2. This is done by dividing the full vector by its length Q.

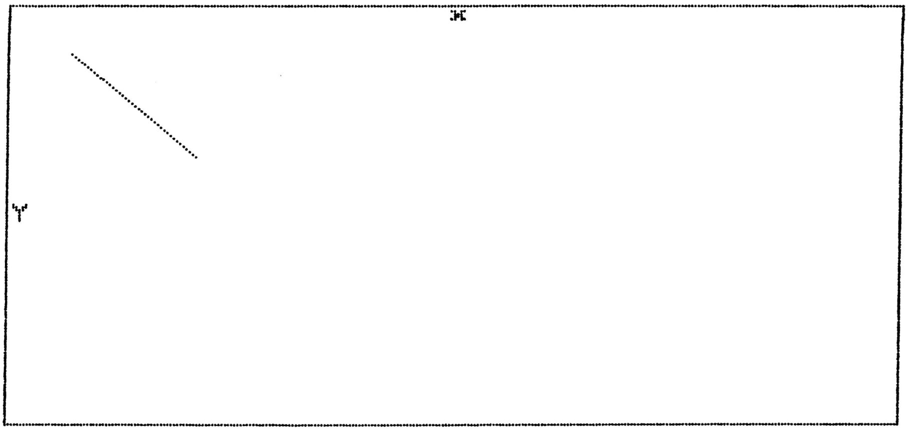
Line 180 starts plotting the points at point 1. Loop 190-230 advances along the line. L is the distance from 1 to the point being plotted. D is the point spacing. Lines 200 and 210 calculate the coordinates of each point being plotted. Line 220 plots the points using the familiar HPLOT x,y command.

```
10  PRINT "PLOT17"
20  PRINT "DRAWS A LINE OF POINTS"
30  PRINT "BETWEEN TWO ARBITRARY"
35  PRINT "BEGINNING AND END"
40  PRINT "POINTS WITH COORDINATES"
50  PRINT "SPECIFIED BY THE USER"
60  PRINT "TYPE CONT TO CONTINUE"
70  STOP
80  HGR
90  HCOLOR= 7
```

PLOT17

```
95  GOSUB 9990
100  INPUT "X1,Y1: ";X1,Y1
110  INPUT "X2,Y2: ";X2,Y2
120  INPUT "D: ";D
130 A = X2 - X1
140 B = Y2 - Y1
150 Q =  SQR (A * A + B * B)
160 UX = A / Q
170 UY = B / Q
180  HPLOT X1,Y1
190  FOR L = D TO Q STEP D
200 XP = X1 + UX * L
210 YP = Y1 + UY * L
220  HPLOT XP,YP
230  NEXT L
240  GOTO 9998
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 4,80 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

```
X1,Y1: 20,20
X2,Y2: 60,60
D: 1
```



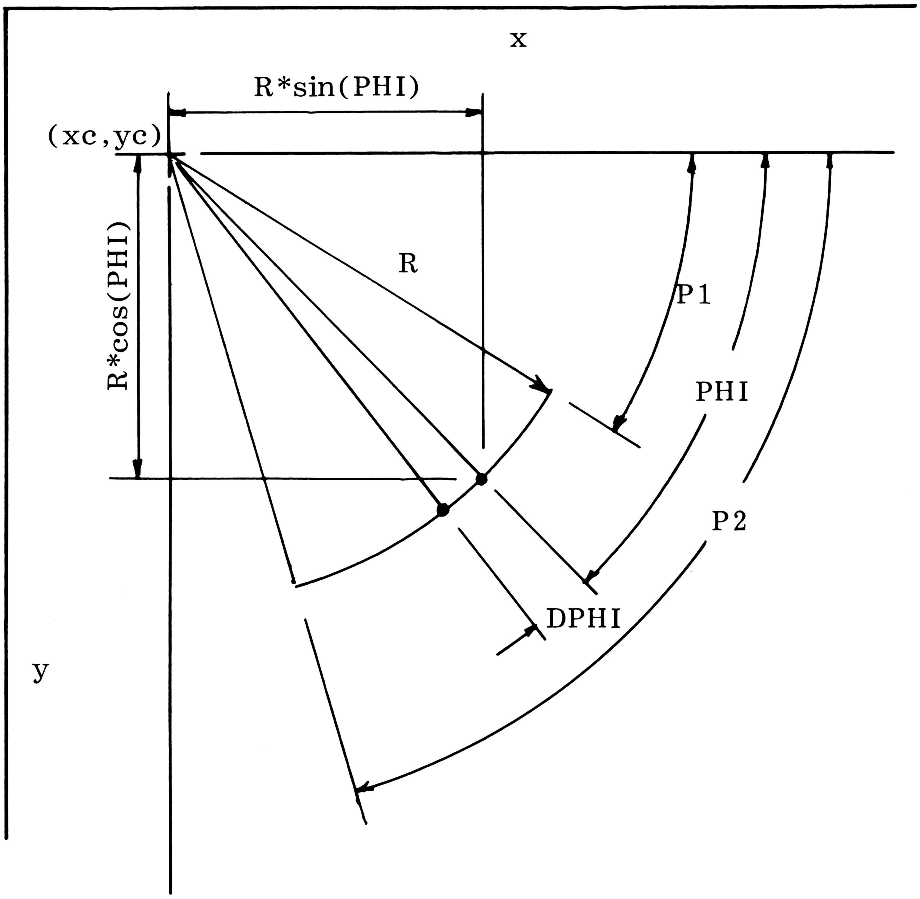
PROGRAM *PLOT60*

PLOT60

This program draws a circle using points. It illustrates the use of trig functions to draw circular shapes.

This program follows the same methodology as *PLOT17* in that points, which are a certain spacing apart, are plotted from a start to an end point. The position of each point is calculated before it is plotted. Whereas in *PLOT17* the calculated positions followed a straight line, here they follow a circular arc. Instead of incrementing a linear spacing, we now increment an angular spacing.

Referring to the sketch, the center of the circle is at xc,yc. The circle's radius is R. The position of a point on the circumference of the circle relative to xc,yc is R*sin(PHI), R*cos(PHI) where PHI is an angle measured from a horizontal line parallel to the x axis. P1 and P2 are the start and end points.



Referring to the program listing, the coordinates of the circle's center are input in line 100. The radius is input in line 120. In line 130, the angular spacing between the points, DPHI, is set equal to 2 degrees. Note that trig functions require an argument in radians, not degrees, so we must convert DPHI from degrees to radians in the program. We could input DPHI in radians and eliminate the conversion, but this is cumbersome. In future programs, all angular quantities will be input in degrees and converted to radians within the program. The conversion is carried out in line 130 where we multiply DPHI by 3.142/180 which is the approximate number of radians in a degree.

Lines 140 and 150 set the start and end angles to 0 and 2*3.142 radians respectively. This will trace out a full circle. Note that we could produce an arc with different values of P1 and P2. These could be input from the keyboard.

Loop 160-220 increments the angle PHI from P1 to P2 in steps DPHI. Lines 170 and 180 calculate the coordinates of each point relative to the center of the circle. Lines 190 and 200 produce global coordinates by adding the coordinates of the center, xc,yc.

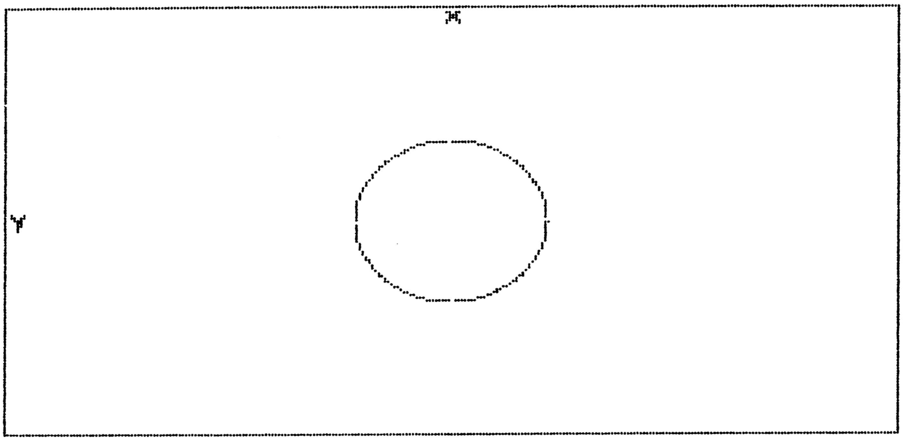
The points are plotted in line 210 using the HPLOT x,y command. Note that we have added in the border plot developed in program *PLOT14*. This is contained in subroutine 9990 which is called from line 75.

```
10  PRINT "PLOT60"
20  PRINT "DRAWS A CIRCLE USING POINTS"
30  PRINT "CENTER COORDINATES AND"
40  PRINT "RADIUS MAY BE SPECIFIED"
50  PRINT "TYPE CONT TO CONTINUE"
60  STOP
70  HGR
80  HCOLOR= 7
85  GOSUB 9990
90  PRINT "INPUT CENTER COORDINATES"
100  INPUT "XC,YC= ";XC,YC
110  PRINT "INPUT RADIUS"
120  INPUT "R= ";R
130 DPHI = 2 * 3.142 / 180
140 P1 = 0
150 P2 = 2 * 3.142
160  FOR PHI = P1 TO P2 STEP DPHI
170 XX = R *  COS (PHI)
180 YY = R *  SIN (PHI)
190 X = XC + XX
200 Y = YC + YY
210  HPLOT X,Y
220  NEXT PHI
```

PLOT60
PLOT61

```
230  GOTO 9998
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

```
INPUT CENTER COORDINATES
XC,YC= 140,80
INPUT RADIUS
R= 30
```



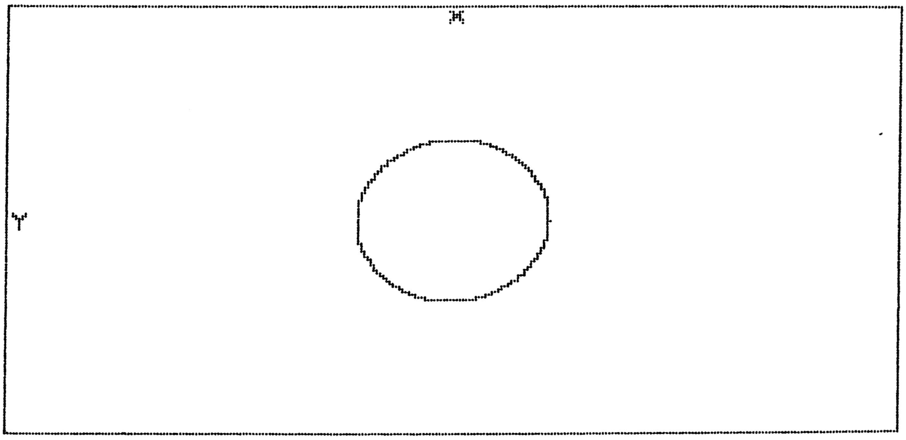
PROGRAM *PLOT61*

Like *PLOT60,* this program also draws a circle but, instead of placing points around the circle's circumference, it uses lines. The beginning an end points of each line segment are calculated exactly as before. When advancing from one point to the next, the program draws a line.

Lines 145 and 146 set the start points for the plotting. Coordinates relative to the circle's center are evaluated in lines 160 and 170. Global coordinates are calculated in lines 180 and 190. Plotting is acomplished in line 200 which employs the HPLOT TO x,y command.

PLOT61

```
10  PRINT "PLOT61"
20  PRINT "DRAWS A CIRCLE USING"
30  PRINT "LINES. CENTER COORDINATES"
40  PRINT "AND RADIUS MAY BE SPECIFIED"
50  PRINT "TYPE CONT TO CONTINUE"
55  STOP
60  HGR
70  HCOLOR= 7
75  GOSUB 9990
80  PRINT "INPUT CENTER COORDINATES"
90  INPUT "XC,YC= ";XC,YC
100  PRINT "INPUT RADIUS"
110  INPUT "R= ";R
120 DPHI = 2 * 3.142 / 180
130 P1 = 0
140 P2 = 2 * 3.142
145 XS = XC + R
146 YS = YC
147  HPLOT XS,YS
150  FOR PHI = P1 TO P2 STEP DPHI
160 XX = R *  COS (PHI)
170 YY = R *  SIN (PHI)
180 X = XC + XX
190 Y = YC + YY
200  HPLOT  TO X,Y
210  NEXT PHI
220  GOTO 9998
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 4,80 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```



PROGRAM *PLOT62*

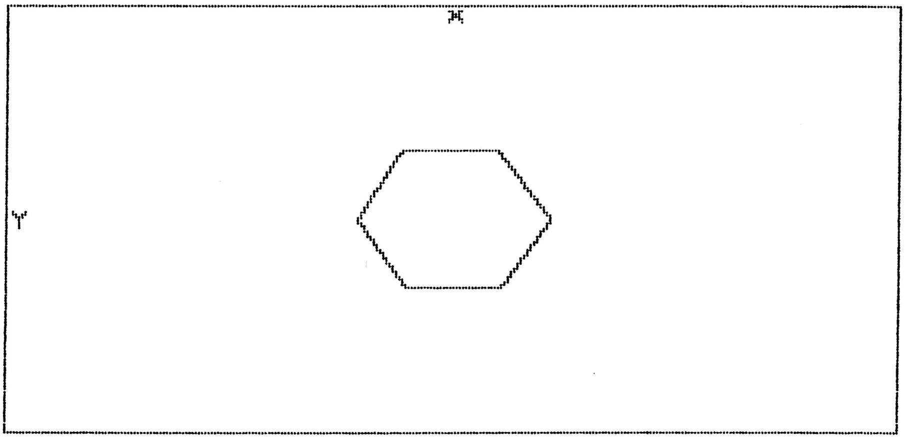
PLOT62

This program is similar to *PLOT61* except that the user may specify the number of sides that the program will draw. This produces a polygon.

In *PLOT61,* the angular increment was set within the program (line 120) to 2 degrees. If we are now to make the number of sides an input option, the angular increment must be calculated within the program. In this program, the number of sides is input in line 125. The corresponding angular increment (in degrees) is calculated in line 126. This is converted to radians in line 127. Plotting continues as before.

```
10  PRINT "PLOT62"
20  PRINT "DRAWS A POLYGON USING"
30  PRINT "LINES. CENTER COORDINATES"
40  PRINT "RADIUS AND NUMBER OF"
45  PRINT "SIDES MAY BE SPECIFIED"
50  PRINT "TYPE CONT TO CONTINUE"
55  STOP
60  HGR
70  HCOLOR= 7
75  GOSUB 9990
80  PRINT "INPUT CENTER COORDINATES"
90  INPUT "XC,YC= ";XC,YC
100  PRINT "INPUT RADIUS"
110  INPUT "R= ";R
120  PRINT "INPUT NUMBER OF SIDES"
125  INPUT "NS= ";NS
➤ 126 DPHI = 360 / NS
127 DPHI = DPHI * 3.14159 / 180
130 P1 = 0
140 P2 = 2 * 3.142
145 XS = XC + R
146 YS = YC
147  HPLOT XS,YS
150  FOR PHI = P1 TO P2 STEP DPHI
160 XX = R *  COS (PHI)
170 YY = R *  SIN (PHI)
180 X = XC + XX
190 Y = YC + YY
200  HPLOT  TO X,Y
210  NEXT PHI
220  GOTO 9998
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 4,80 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

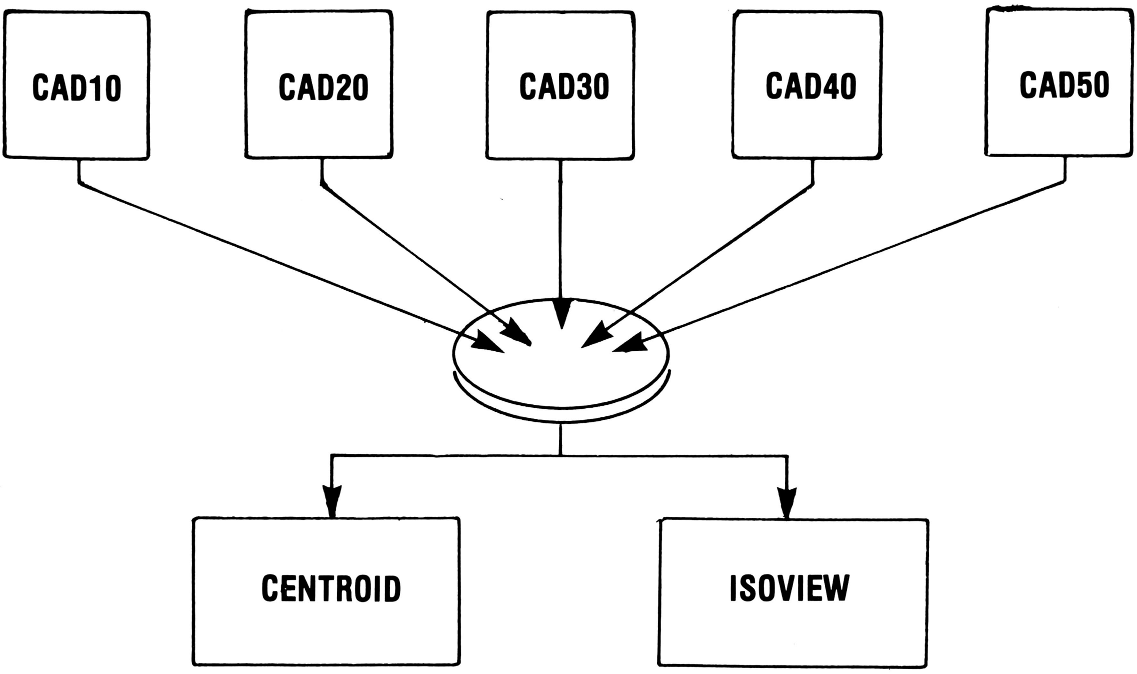
```
]RUN
PLOT62
DRAWS A POLYGON USING
LINES. CENTER COORDINATES
RADIUS AND NUMBER OF
SIDES MAY BE SPECIFIED
TYPE CONT TO CONTINUE

BREAK IN 55
]CONT
INPUT CENTER COORDINATES
XC,YC= 140,80
INPUT RADIUS
R= 30
INPUT NUMBER OF SIDES
NS= 6
ALL DONE
```



Chapter 2: CAD/CAM Software

CAD/CAM Systems, *2-2*
CAD/CAM on Micros, *2-2*
Limitations of Micros, *2-3*
CAD/CAM System Organization, *2-4*
The Drawing Data Base, *2-5*
Real vs Binary Data, *2-5*
Applications Programs, *2-5*
Drawing Revisions, *2-9*
Advantages of CAD/CAM Systems, *2-10*

CAD10:	Interactively creates a 3-dimensional drawing in Apple coordinates and stores on disk, *2-11*
CAD20:	Interactively creates a 3-dimensional drawing in real physical coordinates and stores on disk, *2-19*
CAD30:	Recalls a 3-dimensional drawing from disk, updates it and replaces it on disk, *2-30*
CAD40:	Recalls several 3-dimensional drawings from disk, displays them simultaneously, and positions them on the screen, *2-43*
CAD50:	Recalls a 3-dimensional drawing from disk, prints the drawing data and adds physical properties to members, *2-48*
CENTROID:	Calculates total weight and the centroid location of a 3-dimensional structure and plots the centroid location, *2-58*
ISOVIEW:	Recalls a 3-dimensional drawing from disk and rotates it in 3 dimensions, *2-65*



CAD/CAM SYSTEMS

CAD/CAM is the popular abbreviation for Computer-Aided Design and Computer-Aided Manufacturing. These terms encompass a wide range of technologies and activities relating to the application of computers to design and manufacturing.

During the 1970's many highly successful CAD/CAM systems were developed for use on large main frame computers and small to medium sized mini's. Some of these are exclusively software systems while others incorporate both hardware and software. Their costs range from $50,000 or so up to well over $1,000,000. Many large software systems aren't for sale and can be used on only a service bureau basis. Applications are varied and include tasks such as stress analysis, automated drafting, architectural design, circuit design, and similar operations.

While most of these systems are successful from a technical viewpoint, and are cost/beneficial to their users, they are expensive and unavailable to many people who could use them productively.

CAD/CAM ON MICROS

The remarkable capabilities of our new microcomputers, combined with their low cost, are rapidly changing this situation. Now, serious CAD/CAM work can be carried out on a micro system such as the Apple II or III, the IBMpc, the TRS-80, the HP85 or 87, or the Wang or DECpc's. Any of these systems can be purchased, with very capable peripherals, for less than $10,000.

These systems have splendid graphics capabilities, reasonably fast processing speeds, and a wide variety of peripherals such as color dot matrix printers, multi-color pen plotters, double-sided floppy disks, hard disks, tablets, light pens, digitizers and A/D and D/A converters.

The thought of using a computer like the Apple II for CAD/CAM work sounds great, but the question immediately arises - can it be used for *serious* work? The answer is a definite yes, provided we recognize the limitations of these machines and learn to adapt to them. These computers are much too affordable, accessable, and convenient to use for us to reject them out-of-hand without making a serious effort to adapt our old ways of doing things to accommodate their limitations.

LIMITATIONS OF MICROS

What are the limitations of most micros for CAD/CAM work? There are three: Slow speed, limited active memory and inaccuracies in computations. Regarding speed - I'm not so sure it *does* actually take more time to run a lengthy algorithm, like Monte Carlo optimization, on an Apple II than it does to wait in the queue for access to a busy mainframe. Regarding limited active memories - they're getting bigger all the time. Even if they are limited, we can work around it by writing our software in modular form. Rather than run one big program, we can run several small ones in sequence, storing intermediate data on disk. In fact, that is what we will be doing in this book. Regarding inaccuracies in processing - this is a result of the microprocessor's word size. Some mathematical operations, such as matrix inversion, do require high accuracy in computations. However, usable results can be obtained most of the time with an 8 bit machine. In those rare cases where the accuracy is insufficient, rough answers can always be worked out on a micro and recourse made to a mini or mainframe for the final, high-accuracy solution. This limitation is being resolved by the new generation of 16 bit machines (Wang, DEC, IBMpc). Another generation of 32 bit machines is already on the horizon.

Micros *do* require more human interaction and involvement on the part of the user than larger computers but this should be regarded as a blessing, not a disadvantage. One of the most serious problems with large-computer CAD/CAM systems is that they allow the designer or analyst to use them as a ''black box''. It is seductively easy for him to remove himself from the analytic or design process to much too great an extent. There is too much enticement to simply code the data, throw it into the computer, let the software do its thing, and then believe the results when they come out.

Compare this to the ''old days'' when there were no computers and everything was done by hand. The analyst new what was happening at each step in the process. He had complete interaction with the solution and could spot errors and inconsistencies. He had a ''feeling'' for the problem and could use his creativity and instincts to their fullest. When the human operator is taken out of the loop, a misplaced decimal point, or a wrongly-entered material property can slip by unnoticed with the potential of producing enormous damage. Microcomputers give the operator near-total involvement. They promise to put the analyst and designer, with his judgement, adaptability, creativity, and *common sense* back into the loop where he belongs.

2-4 CAD/CAM SYSTEM ORGANIZATION

Cad/Cam Software

Later in this chapter we will be developing a series of programs that are similar in function to those used by modern large-computer CAD/CAM systems. Figure 2-1 illustrates how a typical system is organized. An INPUT program enables the user to create a picture on the screen of his terminal or computer. This picture, which we shall call a "drawing", may be 2 or 3 dimensional according to the ultimate application. For example, it might by a picture of a piping system on a power plant (3D), or a finite element mesh of a structure (3D), or a printed circuit board layout (2D).

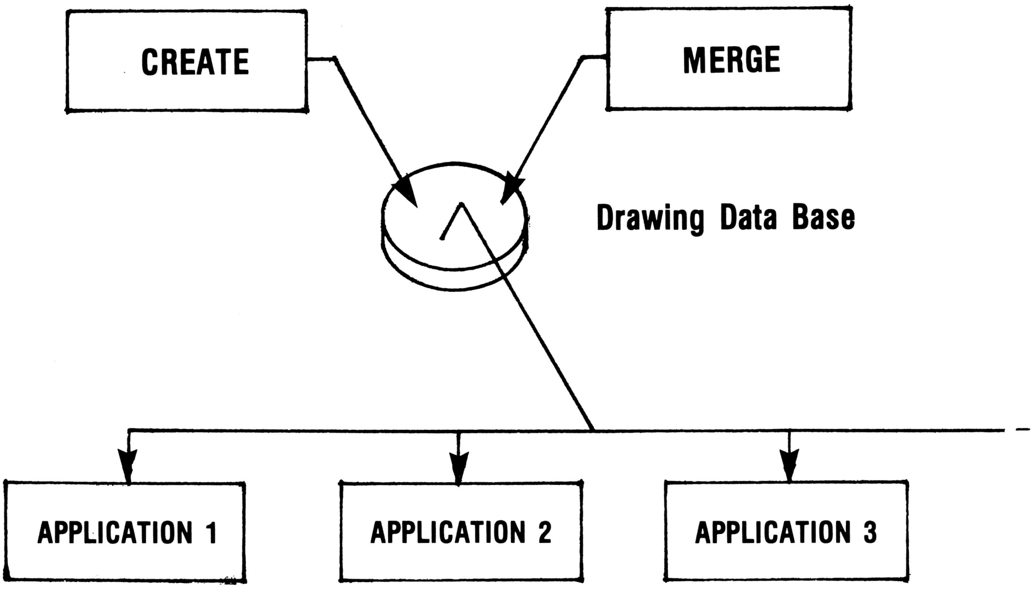

CREATE
MERGE
Drawing Data Base
APPLICATION 1
APPLICATION 2
APPLICATION 3

Figure 2.1: TYPICAL CAD/CAM SOFTWARE SYSTEM

THE DRAWING DATA BASE

We will store the physical data describing the drawing on disk. We will also store all necessary plotting commands so that, when the drawing is recalled from the disk, it may be plotted on the screen. By storing real physical data on the disk, we can recall it and operate on it with various aplications programs.

Let's take a 2 dimensional finite element mesh as an example. Referring to Figure 2-2, suppose we want to carry out a finite element ment stress or heat transfer analysis on the plate shown. We could break it up into a mesh shown having nine node points connected by lines. We would like to store this mesh on disk and recall it later for analysis by either a stress program or a heat transfer program. Or we may want to recall the drawing and modify it by moving a node or redefining a connectivity.

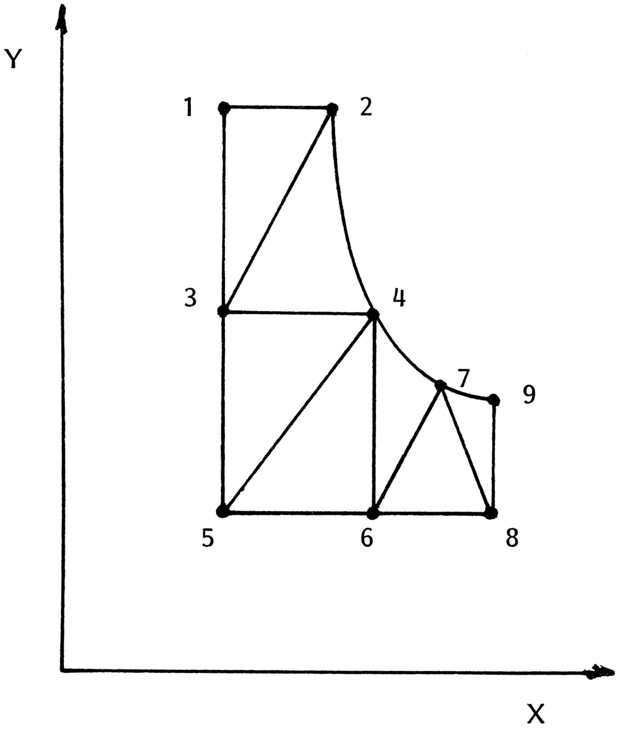
Each node point is defined by a unique number. If we were to store this drawing in the drawing data base we would store the coordinates of the 9 node points - (x1,y1), (x2,y2), ...(x9,y9). We would also store instructions stating which nodes are connected by lines, which nodes define triangular finite elements, physical properties of the elements, etc. We could even define and store the color of the lines for plotting purposes. This is what we mean by a "drawing" which is stored in the drawing data base. It is a file which contains all information necessary to reproduce the drawing on the screen of the computer or on a peripheral plotting device as well as all information required by applications programs which will be operating on the drawing.

REAL VS BINARY DATA

It is important to note that storing real, physical node point coordinates and connectivity information on disk is different from storing a picture as a binary file. Any image on the computer's screen can be stored on disk as a binary file which is just an array of on's and off's. These define which screen pixels are to be turned on (white), and which are to remain off (black). Such a binary file contains no information regarding the physical properties of the drawing and, as a consequence, is useless for further operations. The information we will be storing contains all data required for further operations.

APPLICATIONS PROGRAMS

Referring again to Figure 2-1, the "applications" programs are special programs which carry out various operations on the draw-



```
NP
P1   x1   y1   z1
P2   x2   y2   z2
P3   x3   y3   z3
P4   x4   y4   z4
P5   x5   ...  ...
 .
 .
 .
 .
 .
NL
P1   P2
P1   P3
P2   P3
P3   P4
P2   P4
 .    .
 .    .
 .    .
NT
P1   P2   P3   T1   D1
P2   P3   P4   T2   D2
P3   P4   P5   T3   D3
 .              .
 .              .
 .              .
 .              .
ENDFILE
```

NP=number of node points
Pn=node number
xn,yn,zn=node point coordinates

NL=number of lines
Pn=beginning and end node points

NT=number of triangles
Pn=corner node points
Tn=triangle thickness
Dn=triangle material secific weight

Figure 2.2: FINITE ELEMENT MESH WITH ''ENGINEERING DRAWING'' FILE

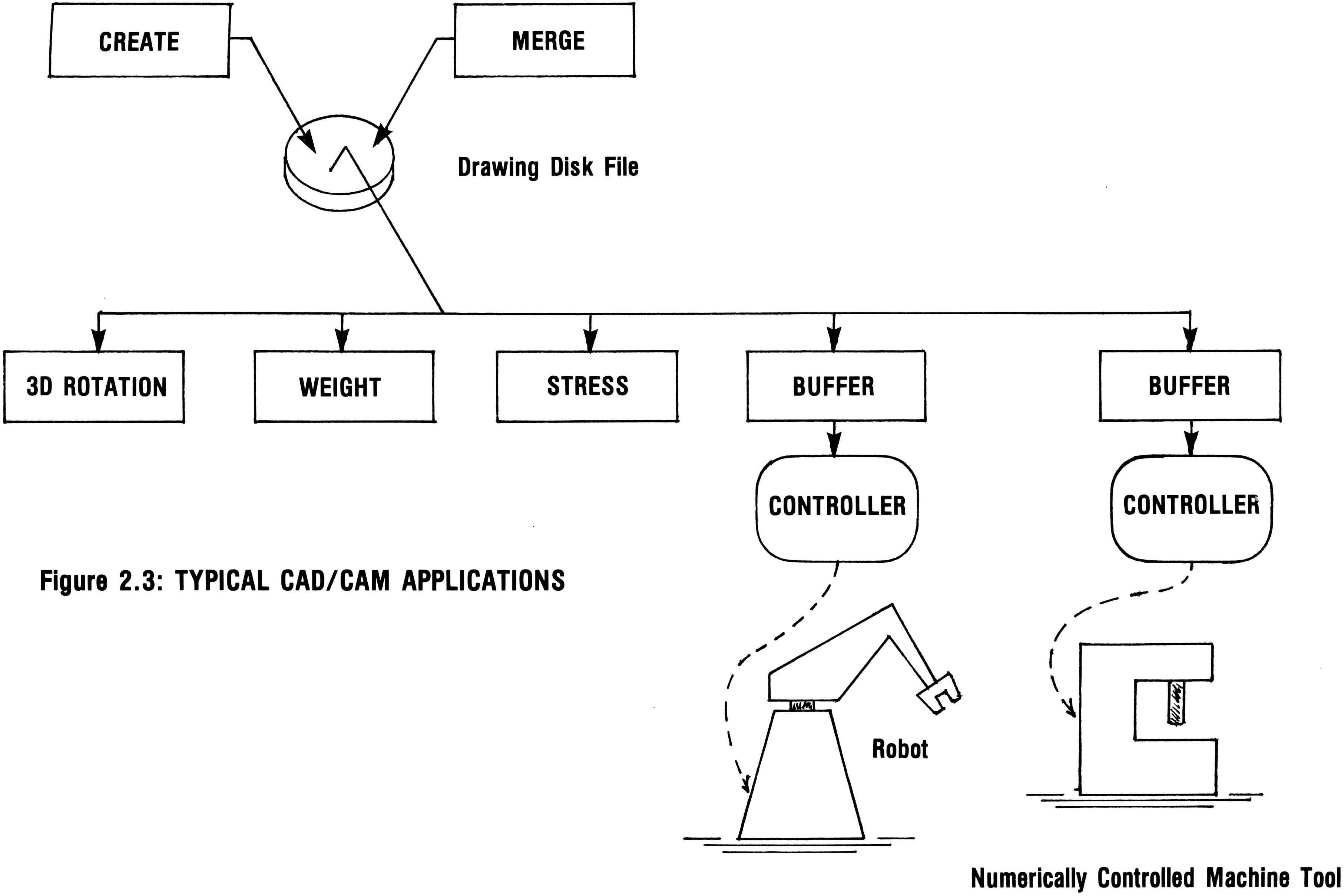

CREATE
MERGE
Drawing Disk File
3D ROTATION
WEIGHT
STRESS
BUFFER
BUFFER
CONTROLLER
CONTROLLER
Robot
Numerically Controlled Machine Tool

Figure 2.3: TYPICAL CAD/CAM APPLICATIONS

ings in the data base. Let's suppose one of them is a program called WEIGHTS which can find the weight of a finite element mesh. WEIGHTS is stored on disk. Let's suppose that the drawing we want to get the weight of is also stored on disk and has the name PLATE.

How would WEIGHT calculate the weight of a finite element mesh? Many schemes could be devised but a simple one would be to calculate the area of each triangular element in the mesh and multiply it by the elements' thickness to get their volume. These would then be multiplied by the material density, which could vary from element to element, to get the element weights. These would be summed to get the total plate weight.

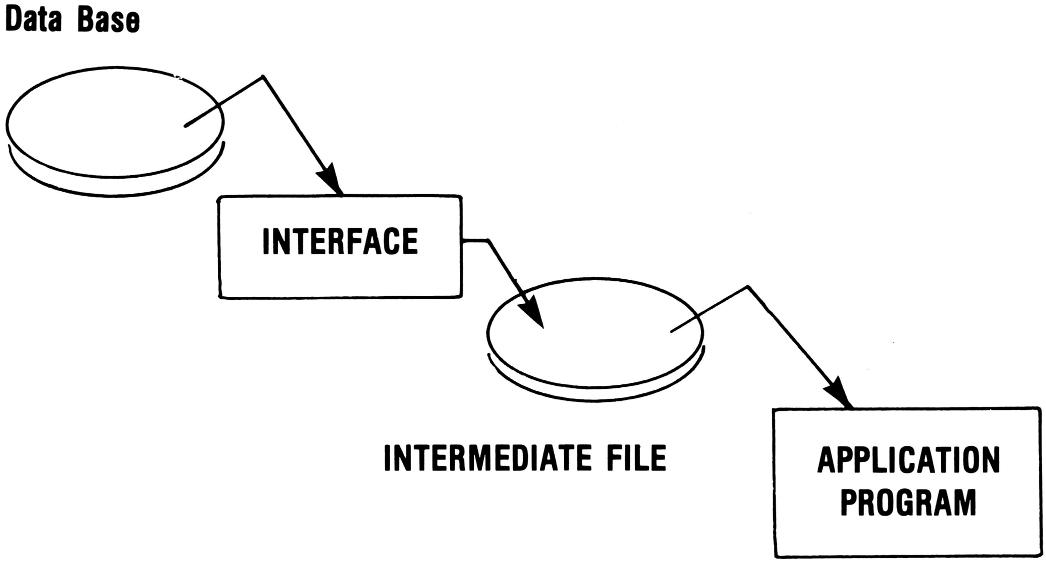
Clearly, if WEIGHT is to carry out these calculations, the file PLATE must contain all necessary information. This would include nodal coordinates, nodes defining elements, element thicknesses and material densities. WEIGHT must also know how this information is stored on disk so it can read it in the proper sequence and format.

It sometimes happens in the development of a complex CAD/CAM system that one group of programmers will design the drawing data base, specifying how and what information is to be stored, while another group writes the applications software. Or there may be several applications programs that have been obtained from different sources and each needs different data in different formats. In such cases either the data base must be reorganized (expensive, time consuming and probably impossible), the aplications programs must be rewritten to conform to the data base (also expensive and time consuming), or interface software must be written. An interface program reads a file from the data base and converts it to the formats and ordering required by the application program. Usually a separate interface program is required for each application program. Often, the interface program writes an intermediate file on tape or disk. This is then read by the application program.

Using this approach, virtually any application program can be interfaced with a drawing data base, provided the data base contains the necessary information. The applications need not be limited to design or analytical operations. They could, for example, be numerically controlled machine tools or robots.

Suppose the application is a machine tool such as a numerically controlled vertical milling machine. The interface program would get a drawing from the data base and convert the contour of a part to be machined to a file of controlling commands required by the mill. This file could be written on disk, magnetic tape, or paper tape. It could be hand carried to the machine, which might be at a different physical location, or it might be read directly by the machine tool, perhaps over a phone line. If we want to look into the future, there is no reason why a robot could not be interfaced to carry out a process, such as 3 dimensional sculpturing.

This is the essence of modern CAD/CAM technology. Whether the computer involved is a mainframe, a mini or a micro, the principles and procedures are fundamentally the same. Data is prepared and stored in a data base. It may then be recalled by an endless array of applications programs to carry out meaningful and useful operations.



DRAWING REVISIONS

One of the most rewarding features of a CAD/CAM system is the ease with which drawings can be revised. For example, in the finite element mesh used as an example earlier, if we want to relocate a point, all we have to do is recall the drawing from disk, redefine the location of the point, and replace the drawing on the disk. The software will take care of readjusting everything else in the drawing and preparing new data files for the applications programs. . Also, new hard copy drawings can be obtained at minimal cost.

ADVANTAGES OF CAD/CAM SYSTEMS

Without the automation afforded by CAD/CAM software, drawing data must be prepared over and over again for each application and for each revision. For example, in the "old days" a draftsman would have drawn a picture of the plate we used as an illustration earlier. He would have run off several paper copies and sent them around to the various departments in the company. The person responsible for weight analysis would have read the data he needed off the drawing and either typed it into a computer or calculated the weight by hand. The person responsible for stress analysis would have read the data he needed off the drawing and typed it into a computer or done the stress calculation by hand. The manufacturing people would have gone throught he same process to get data for their numerically controlled machine tools. And they would have had to repeat this process every time the original drawing was revised!

Many things are wrong with such a system: First, it is error prone. Every time a human reads a bit of data from a drawing, there is an opportunity for an error to slip in. Second, it takes time, which costs money. Third, when a drawing is in a continual state of revision the probability is high that not everyone will be working from the same revision at the same time. Fourth, data extraction is repeated many times which costs money.

In a non-automated system, the primary medium for storing drawing information is paper. In an automated system it is either a magnetic tape, disk or drum. Paper is a great medium for storing information. It is durable, can't be easily erased, and can be read anywhere without special equipment. Its main disadvantage is that it cannot be quickly transmitted. If revisions are being made continually, even a few hours delay in transmitting those changes to the people who need them can create enormous amounts of confusion and waste valuable human resources. With magnetic media, on the other hand, transmission of design changes can be acomplished practically instantaneously.

In the remainder of this chapter, we will be developing a series of programs which can be used to interactively create drawings, attach physical properties to them, store them on disk, recall them, update them, and merge them. We will also develop two applications programs: The first will rotate a 3 dimensional drawing in three dimensions; the second will find the total weight and centroid location of a 3 dimensional structure. The location of the centroid is plotted on the screen.

PROGRAM *CAD10*

This program can be used to interactively create a drawing composed of points and lines. The drawing may then be stored on disk. Coordinates of points comprising the drawing must be in Apple units and in the Apple's screen coordinate system.

There are seven major sections to this program:

A - Control
B - Accepts point data and plots points
C - Out of bounds check on point data
D - Accepts line data and plots lines
E - Finds points which define a line
F - Files drawing on disk
G - Zeros all drawing parameters

Section A is the controlling portion of the program. The program is menu driven and the menu is presented in line 125. This asks whether we want to draw a point (P) or a line (L), or whether we want to file a drawing on disk (FILE) or terminate execution (END). After one of these options is selected, control drops to the proper portion of the program. The loop 125-180 is a continually cycling loop. The only way out is by typing END.

Section B accepts point data and plots the point. The parameter NP in line 3005 keeps track of the number of points input. This will be needed later when the drawing is filed on disk. Line 3010 asks for the number of the point, P(NP). The array P(NP) was dimensioned to 50 in line 80 so we can input up to 50 points. Note, however, that the number we assign a point does not have to be less than 50. For example, P(17) = 9435 says that the seventeenth point input is to be assigned the indentifying number 9435. This is acceptable. Line 3020 asks for the coordinates of the point. Line 3030 transfers control to subroutine 4000 which peforms a simple out of bounds check before attempting to plot the point. The parameter OB in lines 3035 and 3036 is an out of bounds switch. It equals 1 if the point is out of bounds. Otherwise it is 0. It is set in subroutine 4000, which we will discuss shortly. If a point is out of bounds, line 3035 subtracts 1 from the total number of points and line 3036 bypasses the plotting commands. Control then returns to line 125 where another input command is given. If the point is in bounds, line 3040 asks for the color we wish to assign to the point and line 3045 plots it. Note that we could specify a color equal to 0, in which case we would erase the point from the screen. The number 7 draws it in white. If we specify a color equal to 0, the point is erased from the screen, but not from the point file. That is, even though we cannot see it, it is still carried in memory as a point (with color 0) and will be written on the disk file as such.

Section C, the out of bound check on a point, simply checks to see if the x and y coordinates of a point lie within the screen boundaries x = 0 to 279, y = 0 to 159. If not, the proper message is printed and the out of bounds switch, OB, is set equal to 1. Control ultimately returns to line 125 where a new input command is specified. Note that the program could be modified to retain an out of bounds point in the point file, but just not plot it. The problem with this is that, if the point is later used to define the beginning or end of a line, there will be an out of bounds error when the line is plotted unless a clipping algorithm is employed.

Section D defines lines. NL in line 3110 counts the number of lines. The numbers of the beginning and end points are input in line 3120. For example, when we specified the numbers of the points in Section B, we might have assigned the first point input (NP = 1) the number 17 (P(1) = 17) and the fifth point input (NP = 5) the number 23 (P(5) = 23). If we then define a line as running from point 17 to 23, that is, from LB(17) to LE(23), a line will be drawn from x(1),y(1) to x(5),y(5). Since the point coordinates x(I),y(I) are stored in sequential order (1-50), we must find which values of I correspond to the point numbers 17 and 23. This is done in subroutine 4200 which we will explain shortly. The parameter MP in lines 3135 and 3136 is a missing point switch. If subroutine 4200 can't find a particular point, it sets MP = 1 in which case 1 is subtracted from the total number of lines and the plotting instructions are bypassed. Control then returns to line 125 where a new set of commands are entered. If the beginning and end points are found, the switch remains set to 0 and the line is plotted in 3160. Line 3140 asks for the color of the line. Comments made regarding erasing points by setting the color equal to 0 are valid with lines as well.

Section E, which is the previously mentioned subroutine 4200, finds the coordinates of the points defining the beginning and end of a line. Suppose the points had been input in the following sequence:

I = 1	P(1) = 17	I = A	↓	Search for points 17 and 23. These correspond to locations I = 1 and I = 5.
I = 2	P(2) = 13			
I = 3	P(3) = 4			
I = 4	P(4) = 9			
I = 5	P(5) = 23	I = B		

Suppose also that we define the 18th line by LB(18) = 17 and LE(18) = 23. Loop 4210-4240 searches through the P(I) array until it finds the beginning node of line 18; LB(18) = 17. This corresponds to I = 1 in this example. When it finds node 17 it sets A equal to the corresponding value of I, which is 1 here. Control then drops to line 4270 where the same process is repeated for the end node, LE(18) = 23. If either node can't be found in the P(I) array, the message "CAN'T FIND POINT - " is printed; 1 is subtracted from the total number of lines; MP is turned on, and control returns to line 125.

A and B are the positions in the P(I) array of the beginning and end points. These define which coordinates correspond to points 17 and 23. Line 3160 draws a line between these two points.

The array P(I) serves as a form of "pointer" in that it points to the position in the coordinate array corresponding to a particular node:

XP(1)	YP(1)	ZP(1)	A = 1 ← P(1) = 17
XP(2)	YP(2)	ZP(2)	P(2) = 13
XP(3)	YP(3)	ZP(3)	P(3) = 4
XP(4)	YP(4)	ZP(4)	P(4) = 9
XP(5)	YP(5)	ZP(5)	B = 5 ← P(5) = 23

```
3160  HPLOT XP(A),YP(A) TO XP(B),YP(B)
```

Subroutine 4800 writes the drawing to disk. Line 4860 asks for the name we wish to give the drawing. Lines 4880-4900 open a sequential disk file. Line 4870 asks for the drive number. If you are not familiar with opening and closing disk files on the Apple computer, please see the DOS3.3 manual, Chapter 6, for a full discussion. Line 4880 defines the parameter D$ as a CTRL-D. That is, when typing this line into your computer, type D$ = " and then depress the CTRL and the D keys simultaneously. Then type the closed quote, ".

Lines 4910-5040 write the point and line information on the disk file. Note that we are writing the actual point coordinates along with the line information. When we later recall this file, we can replot the drawing, change it, or do anything else we want with it since we have all the coordinate information available. Note also that when we recall this drawing we will have to read exactly the same number of points and lines and in exactly the same sequence as they are written on the file. This is the reason why we write the number of points and lines, NP and NL, at the beginning of the file.

CAD10

Line 5105 closes the file. Line 5106 clears the screen. Line 5107 redraws the border around the plotting area by calling subroutine 9990. 5108 zeros all drawing variables in preparation for the creation of a new drawing.

```
10  PRINT "CAD10"
20  PRINT "CREATES A 3D DRAWING"
30  PRINT "TYPE CONT TO CONTINUE"
40  STOP
50  HGR
60  HCOLOR= 7
70  GOSUB 9990
80  DIM P(50),XP(50),YP(50),ZP(50),PC(50)
90  DIM LB(25),LE(25),LC(25)
106  REM
107  REM
108  REM
109  REM
110  REM  INPUT
125  INPUT "P,L,FILE OR END: ";CK$
130  IF CK$ = "P" THEN  GOSUB 3000
140  IF CK$ = "L" THEN  GOSUB 3100
160  IF CK$ = "FILE" THEN  GOSUB 4800
170  IF CK$ = "END" THEN  GOTO 9998
180  GOTO 125
2996  REM
2997  REM
2998  REM
2999  REM
3000  REM  POINT INPUT
3005 NP = NP + 1
3010  INPUT "POINT NO: ";P(NP)
3020  INPUT "X,Y,Z COORDS: ";XP(NP),YP(NP),ZP(NP)
3030  GOSUB 4000
3035  IF OB = 1 THEN NP = NP - 1
3036  IF OB = 1 GOTO 3050
3040  INPUT "COLOR: ";PC(NP)
3044  HCOLOR= PC(NP)
3045  HPLOT XP(NP),YP(NP)
3050  RETURN
3096  REM
3097  REM
```

A - Control (lines 110–180)

B - Accepts point data and plots points (lines 3000–3050)

C - Out of bounds check on data points

```
3098  REM
3099  REM
3100  REM  LINE INPUT
3110 NL = NL + 1
3120  INPUT "BEGIN/END PTS ";LB(NL),LE(NL)
3130  GOSUB 4200
3135  IF MP = 1 THEN NL = NL - 1
3136  IF MP = 1 GOTO 3170
3140  INPUT "COLOR: ";LC(NL)
3150  HCOLOR= LC(NL)
3160  HPLOT XP(A),YP(A) TO XP(B),YP(B)
3170  RETURN
3996  REM
3997  REM
3998  REM
3999  REM
```

D - Accepts line data and plots lines

```
4000  REM  OUT OF BOUNDS CHECK
4005 OB = 0
4010  IF XP(NP) < 0 GOTO 4100
4020  IF XP(NP) > 279 GOTO 4110
4030  IF YP(NP) < 0 GOTO 4120
4040  IF YP(NP) > 159 GOTO 4130
4050  RETURN
4100  PRINT "XP<0-TRY AGAIN"
4105  GOTO 4135
4110  PRINT "XP>279-TRY AGAIN"
4115  GOTO 4135
4120  PRINT "YP<0-TRY AGAIN"
4125  GOTO 4135
4130  PRINT "YP>159-TRY AGAIN"
4135 OB = 1
4140  RETURN
4196  REM
4197  REM
4198  REM
4199  REM
```

E - Finds points which define a line

```
4200  REM  FIND POINTS
4205 MP = 0
4210  FOR I = 1 TO NP
4220  IF LB(NL) = P(I) THEN A = I
4230  IF LB(NL) = P(I) GOTO 4270
4240  NEXT I
4250  PRINT "CAN'T FIND POINT ";LB(NL)
4255 MP = 1
4260  GOTO 4330
4270  FOR I = 1 TO NP
4280  IF LE(NL) = P(I) THEN B = I
4290  IF LE(NL) = P(I) GOTO 4330
4300  NEXT I
4310  PRINT "CAN'T FIND POINT ";LE(NL)
4315 MP = 1
4330  RETURN
4796  REM
4797  REM
```

CAD10

```
4798  REM
4799  REM
```

F - Files a drawing on disk

```
4800  REM  WRITE DATA TO DISK
4860  INPUT "FILE NAME: ";F$
4870  INPUT "DRIVE NUMBER: ";DR
4880 D$ = "": REM  CTRL-D
4890  PRINT D$;"OPEN";F$;",D";DR
4900  PRINT D$;"WRITE";F$
4910  PRINT NP
4920  FOR I = 1 TO NP
4930  PRINT P(I)
4940  PRINT XP(I)
4950  PRINT YP(I)
4960  PRINT ZP(I)
4970  PRINT PC(I)
4980  NEXT I
4990  PRINT NL
5000  FOR I = 1 TO NL
5010  PRINT LB(I)
5020  PRINT LE(I)
5030  PRINT LC(I)
5040  NEXT I
5105  PRINT D$;"CLOSE"
5106  HGR
5107  GOSUB 9990
5108  GOSUB 5200
5110  RETURN
5196  REM
5197  REM
5198  REM
5199  REM
```

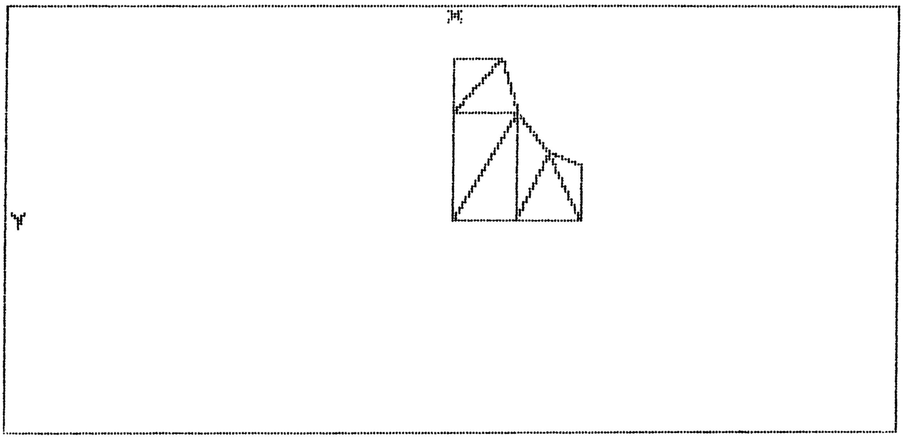
G - Zeros all drawing parameters

```
5200  REM  ZERO ARRAYS
5210  FOR K = 1 TO 50
5220 P(K) = 0
5230 XP(K) = 0
5240 YP(K) = 0
5250 ZP(K) = 0
5260 PC(K) = 0
5270  NEXT K
5280  FOR K = 1 TO 25
5290 LB(K) = 0
5300 LE(K) = 0
5310 LC(K) = 0
5320  NEXT K
5330 NP = 0
5340 NL = 0
5350  RETURN
9986  REM
9987  REM
```

```
9988  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

SAMPLE RUN

As an illustration of how this program is used, we'll draw the framework shown in Figure 2-2. Notice that we *won't* number the nodes in consecutive numerical order. Note also that we are using Apple plotting units and the Apple's screen coordinate system to define the node locations.



```
P,L,FILE OR END: P
POINT NO: 1
X,Y,Z COORDS: 140,20,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 3
X,Y,Z COORDS: 140,40,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 5
X,Y,Z COORDS: 140,80,0
COLOR: 7
P,L,FILE OR END: P
```

```
POINT NO: 6
X,Y,Z COORDS: 160,80,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 8
X,Y,Z COORDS: 180,80,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 9
X,Y,Z COORDS: 380,60,0      ←  Note error
XP>279-TRY AGAIN
P,L,FILE OR END: P
POINT NO: 9
X,Y,Z COORDS: 180,60,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 7
X,Y,Z COORDS: 170,55,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 4
X,Y,Z COORDS: 160,40,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 2
X,Y,Z COORDS: 155,20,0
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 1,3
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 3,55          ←  Note error
CAN'T FIND POINT 55
P,L,FILE OR END: L
BEGIN/END PTS 3,5
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 5,6
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 6,8
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 8,9
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 9,77
CAN'T FIND POINT 77         ←  Note error
P,L,FILE OR END: L
BEGIN/END PTS 9,7
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 7,4
```

```
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 4,2
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 1,2
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 3,2
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 3,4
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 4,5
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 4,6
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 6,7
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 7,8
COLOR: 7
P,L,FILE OR END: FILE
FILE NAME: TRUSS
DRIVE NUMBER: 1
P,L,FILE OR END: END
ALL DONE
```

PROGRAM *CAD20*

In program *CAD10* we were able to create and store drawings, but the input data had to be in Apple units and in the Apple's coordinate system. That is, values of x had to lie between 0 and 279 and y between 0 and 159. The x axis had to run from left to right across the top of the screen; the y axis from top to bottom down the left side.

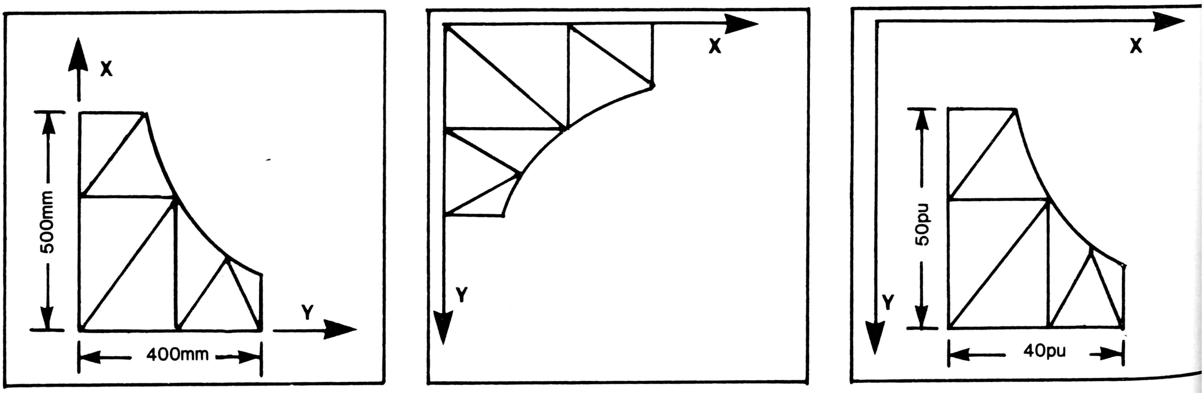
This, of course, is not very convenient if we want to create a drawing using real physical units in the coordinate system of our choice. For example, we might want to type in a drawing of a machine part that had units in millimeters with values exceeding 279. Also, the coordinate system on the drawing might have its coordinate axes oriented in a more conventional manner with x running from left to right across the bottom and y running up the left side.

We could always manually scale the coordinates to fit on the Apple's screen and redefine the drawing's coordinate system, but that would be much to cumbersome. What we will do is modify program *CAD10* to accept data of any size and in any arbitrary coordinate system.

CAD20

This program, *CAD20,* is very similar to *CAD10* in that you can draw, erase and store points and lines. An additional feature, however, is that the input data may be of any size and referred to any coordinate system. *CAD20* will scale the data to fit on the screen and will also offset and rotate the data to conform to any arbitrary coordinate system.

For example, referring to the three sketches in **Figure 2.4** the first shows a drawing in real physical units and in a coordinate system that we might expect to find on a paper engineering drawing. The second shows how it would look if scaled to fit on the Apple's screen, with no coordinate transformation. The third shows how we would *like* it to appear on the screen. Our job is to write program *CAD20* so that the user can take data points directly off a real drawing (sketch 1) and have them plotted on the screen in the same orientation (sketch 3). In this example, 500 millimeters would become 50 plotting units; the original coordinate system would be rotated 90 degrees clockwise, and all plotting values would be offset a certain amount so that the 0,0 point does not appear in the upper left corner of the screen.



(1) real physical drawing before plotting

(2) drawing scaled and plotted

(3) drawing scaled, rotated, offset and plotted

Figure 2.4: TRANSFORMATION FROM REAL PHYSICAL COORDINATES AND UNITS TO APPLE PLOTTING SYSTEM

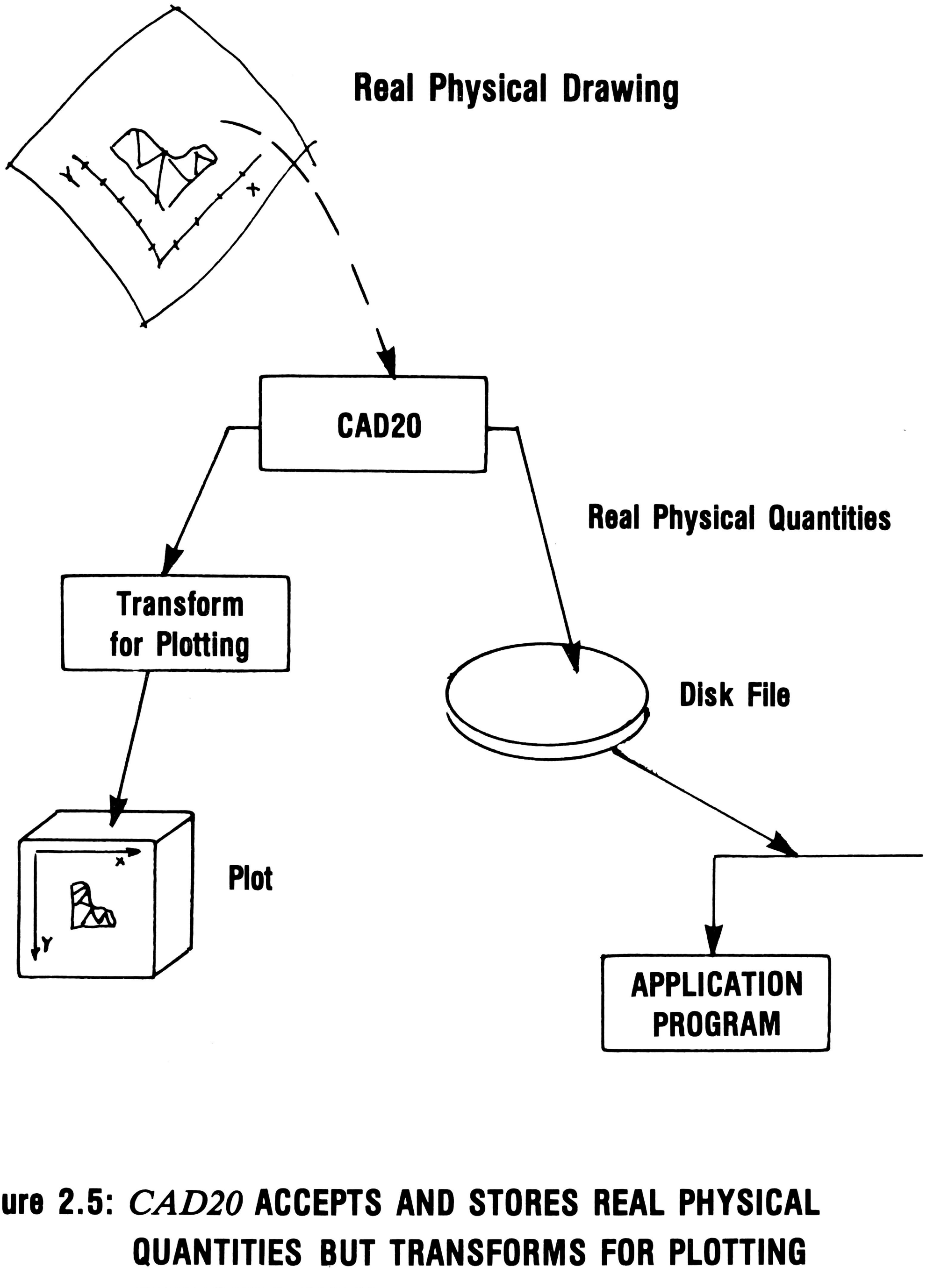
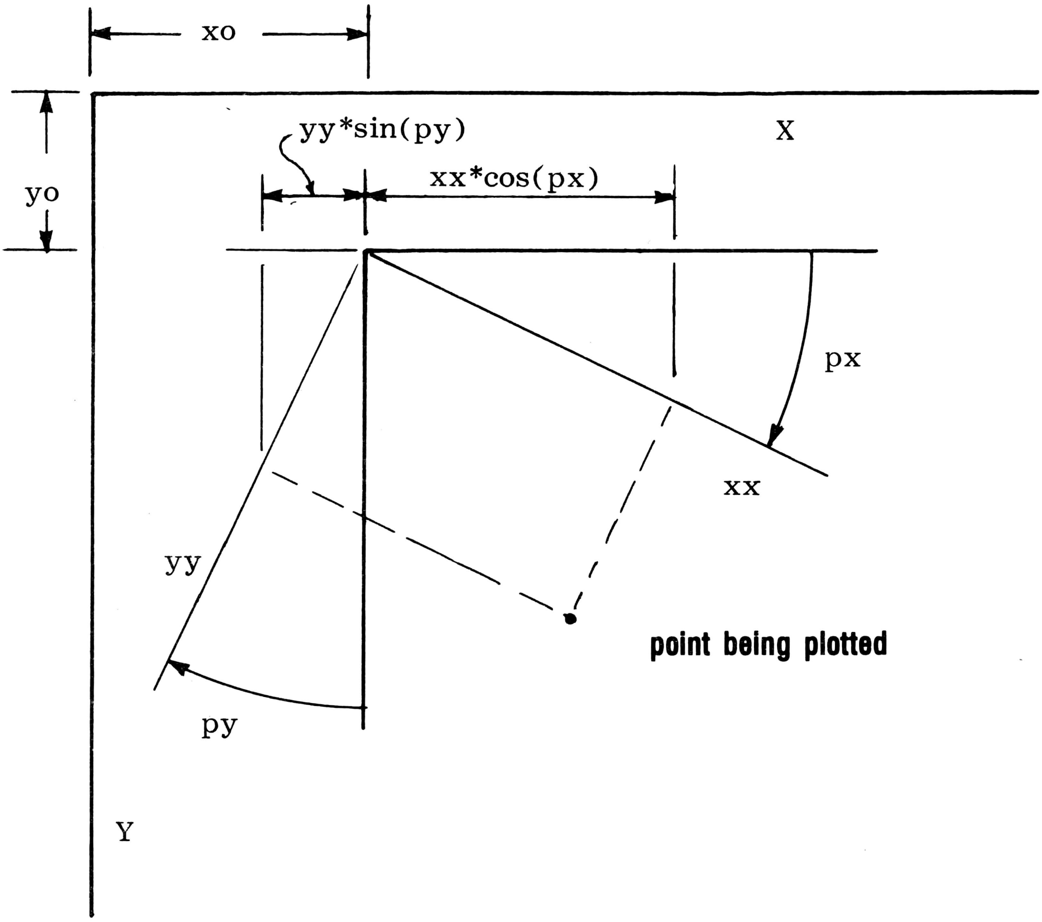

Real Physical Drawing
Y
X
CAD20
Real Physical Quantities
Transform for Plotting
Disk File
X
Y
Plot
APPLICATION PROGRAM

Figure 2.5: *CAD20* ACCEPTS AND STORES REAL PHYSICAL QUANTITIES BUT TRANSFORMS FOR PLOTTING PURPOSES ONLY.

CAD20 will do this for us automatically, but we must first tell it what the scale factor is to be, what the new orientation of the axes is relative to the Apple's axes, and what the offset values are to be. The transformation is then carried out according to the following analysis:

COORDINATE TRANSFORMATION

The sketch below shows a set of real coordinate axes (xx,yy) relative to the Apple's axes (x,y). xo,yo is the location of the origin of the real axes relative to the Apple's axes in Apple plotting units. PX is the *clockwise* angle of the xx axis relative to the x axis; PY is the *clockwise* angle of the yy axis relative to the y axis.



A set of real physical coordinates is related to screen plotting units by the following two equations:

$$x = xo + SF*[xx*\cos(PX)-yy*\sin(PY)]$$

$$y = yo + SF*[xx*\sin(PX) + yy*\cos(PY)]$$

where SF is the user defined scale factor and xo,yo are the user defined offset values.

The term xx*cos(PX) - yy*sin(PY) produces the real value of x in the Apple's coordinate system, rotated, but unscaled and un shifted. Multiplying that quantity by SF scales it to fit on the screen; adding xo offsets it to a convenient position on the screen. Similarly for the y coordinates.

The angles PX and PY *must* be defined as shown. The scale factor SF must be predetermined by hand. For example, suppose the drawing has a maximum range of values of 500 millimeters and we want it to fit within 180 plotting units. The scale factor would be:

$$SF = 180/500 \text{ plotting units/millimeter}$$

PROGRAM LISTING

The listing of program *CAD20* is very similar to *CAD10* with the addition of the subroutines starting at lines 5300 and 5400. Subroutine 5300 asks for SF, PX, PY, xo and yo. These are used in the transformation equations. Subroutine 5400 implements the transformation equations and scales, rotates and offsets each point before it is plotted. This subroutine is called by lines 3047, 3157 and 3162 which appear just prior to plotting commands. In other words, before a point is plotted, it is transformed by subroutine 5400.

It is important to note that the original physical values of the plot points are not changed. Subroutine 5400 operates on dummy variables xr and yr and converts these to transformed dummy variables xt and yt. The original physical coordinates xp(I), yp(I), zp(I) are not changed.

Looking at lines 3000-3070, for example, we see that the real physical coordinates are input in line 3020. Lines 3045 and 3046 set xr and yr equal to these quantities. Line 3047 calls subroutine 5400 which transforms xr and yr, returning the quantities xt and yt. Line 3048 calls subroutine 4000 which performs an out of bounds check on xt and yt. These quantities are plotted in line 3060.

As in *CAD10,* subroutine 4800 files the drawing data on disk. Note that the real physical values are filed, not the transformed plot values (see Figure 2-5). When other programs recall this drawing from the disk, they must retransform the data prior to plotting. Note that we are including the transformation parameters in the disk file in lines 4905-4909.

```
10  PRINT "CAD20"
20  PRINT "CREATES A 3D DRAWING"
21  PRINT "FROM REAL PHYSICAL COORDS"
22  PRINT "SCALES, ROTATES AND OFFSETS"
23  PRINT "PLOT COORDINATES"
30  PRINT "TYPE CONT TO CONTINUE"
40  STOP
50  HGR
60  HCOLOR= 7
70  GOSUB 9990
75  GOSUB 5300
80  DIM P(50),XP(50),YP(50),ZP(50),PC(50)
90  DIM LB(40),LE(40),LC(40)
106  REM
107  REM
108  REM
109  REM
110  REM  INPUT
125  INPUT "P,L,FILE OR END: ";CK$
130  IF CK$ = "P" THEN  GOSUB 3000
140  IF CK$ = "L" THEN  GOSUB 3100
150  IF CK$ = "FILE" THEN  GOSUB 4800
160  IF CK$ = "END" GOTO 9998
170  GOTO 125
2996  REM
2997  REM
2998  REM
2999  REM
3000  REM  POINT INPUT
3005 NP = NP + 1
3010  INPUT "POINT NO: ";P(NP)
3020  INPUT "X,Y,Z COORDS: ";XP(NP),YP(NP),ZP(NP)
3040  INPUT "COLOR: ";PC(NP)
3044  HCOLOR= PC(NP)
3045 XR = XP(NP)
3046 YR = YP(NP)
```

```
➤ 3047  GOSUB 5400
  3048  GOSUB 4000
  3050  IF OB = 1 THEN NP = NP - 1
  3055  IF OB = 1 GOTO 3070
  3060  HPLOT XT,YT
  3070  RETURN
  3096  REM
  3097  REM
  3098  REM
  3099  REM
  3100  REM  LINE INPUT
  3110 NL = NL + 1
  3120  INPUT "BEGIN/END PTS ";LB(NL),LE(NL)
  3125  GOSUB 4200
  3126  IF MP = 1 THEN NL = NL - 1
  3127  IF MP = 1 GOTO 3170
  3130  INPUT "COLOR: ";LC(NL)
  3150  HCOLOR= LC(NL)
  3155 XR = XP(A)
  3156 YR = YP(A)
➤ 3157  GOSUB 5400
  3158 AA = XT
  3159 BB = YT
  3160 XR = XP(B)
  3161 YR = YP(B)
➤ 3162  GOSUB 5400
  3163 CC = XT
  3164 DD = YT
  3165  HPLOT AA,BB TO CC,DD
  3170  RETURN
  3996  REM
  3997  REM
  3998  REM
  3999  REM
  4000  REM  OUT-OF-BOUNDS CHECK
  4005 OB = 0
  4010  IF XT < 0 GOTO 4100
  4020  IF XT > 279 GOTO 4110
  4030  IF YT < 0 GOTO 4120
  4040  IF YT > 159 GOTO 4130
  4050  GOTO 4150
  4100  PRINT "XT<0-TRY AGAIN"
  4105  GOTO 4140
  4110  PRINT "XT>279-TRY AGAIN"
  4115  GOTO 4140
  4120  PRINT "YT<0-TRY AGAIN"
  4125  GOTO 4140
  4130  PRINT "YT>0-TRY AGAIN"
  4140 OB = 1
  4150  RETURN
  4196  REM
  4197  REM
  4198  REM
  4199  REM
  4200  REM  FIND POINTS
  4205 MP = 0
  4210  FOR I = 1 TO NP
  4220  IF LB(NL) = P(I) THEN A = I
  4230  IF LB(NL) = P(I) GOTO 4270
  4240  NEXT I
  4250  PRINT "CAN'T FIND POINT ";LB(NL)
  4255 MP = 1
  4260  GOTO 4330
  4270  FOR I = 1 TO NP
  4280  IF LE(NL) = P(I) THEN B = I
  4290  IF LE(NL) = P(I) GOTO 4330
  4300  NEXT I
  4310  PRINT "CAN'T FIND POINT ";LE(NL)
  4315 MP = 1
  4330  RETURN
  4800  REM  WRITE DATA TO DISK
  4820  IF CK$ = "Y" GOTO 4860
  4830  IF CK$ = "N" GOTO 9998
  4860  INPUT "FILE NAME: ";F$
  4870  INPUT "DRIVE NUMBER: ";DR
  4880 D$ = "": REM  CTRL-D
  4890  PRINT D$;"OPEN";F$;",D";DR
  4900  PRINT D$;"WRITE";F$
➤ 4905  PRINT SF
➤ 4906  PRINT XO
➤ 4907  PRINT YO
➤ 4908  PRINT PX
➤ 4909  PRINT PY
  4910  PRINT NP
  4920  FOR I = 1 TO NP
  4930  PRINT P(I)
  4940  PRINT XP(I)
  4950  PRINT YP(I)
  4960  PRINT ZP(I)
  4970  PRINT PC(I)
  4980  NEXT I
  4990  PRINT NL
  5000  FOR I = 1 TO NL
  5010  PRINT LB(I)
  5020  PRINT LE(I)
  5030  PRINT LC(I)
  5040  NEXT I
  5105  PRINT D$;"CLOSE"
  5106  HGR
  5107  GOSUB 9990
  5108  GOSUB 5200
  5110  RETURN
  5196  REM
  5197  REM
```

```
5198  REM
5199  REM
5200  REM  ZERO ARRAYS
5205  FOR K = 1 TO 50
5210 P(K) = 0
5215 XP(K) = 0
5220 YP(K) = 0
5225 ZP(K) = 0
5230 PC(K) = 0
5235  NEXT K
5240  FOR K = 1 TO 25
5245 LB(K) = 0
5250 LE(K) = 0
5255 LC(K) = 0
5260  NEXT K
5265 NP = 0
5270 NL = 0
5280  RETURN
5300  REM  SCALE,ROTATE AND OFFSET INPUT COORDS
5305  INPUT "SF: ";SF
5306  INPUT "XO,YO (APPLE UNITS): ";XO,YO
5310  INPUT "PX: ";PX
5320  INPUT "PY: ";PY
5330 PX = PX * 3.14159 / 180
5340 PY = PY * 3.14159 / 180
5350  RETURN
5396  REM
5397  REM
5398  REM
5399  REM
5400  REM  SCALE,ROTATE AND OFFSET PLOT COORDS
5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
9985  REM
9986  REM
9987  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

SAMPLE RUN

The truss shown in Figure 2-6a has units in millimeters with the x axis vertical and the y axis horizontal. If this drawing were to be input directly into either *CAD10* or *CAD20,* even if the coordinate values were scaled down to fit on the screen the picture would come out rotated 90 degrees clockwise. We would like it to appear as shown in Figure 2-6b.

To get 2-6b, we use *CAD20* with the following values:

CAD20

SF = .2
xo = 100
yo = 130
PX = 270
PY = 270

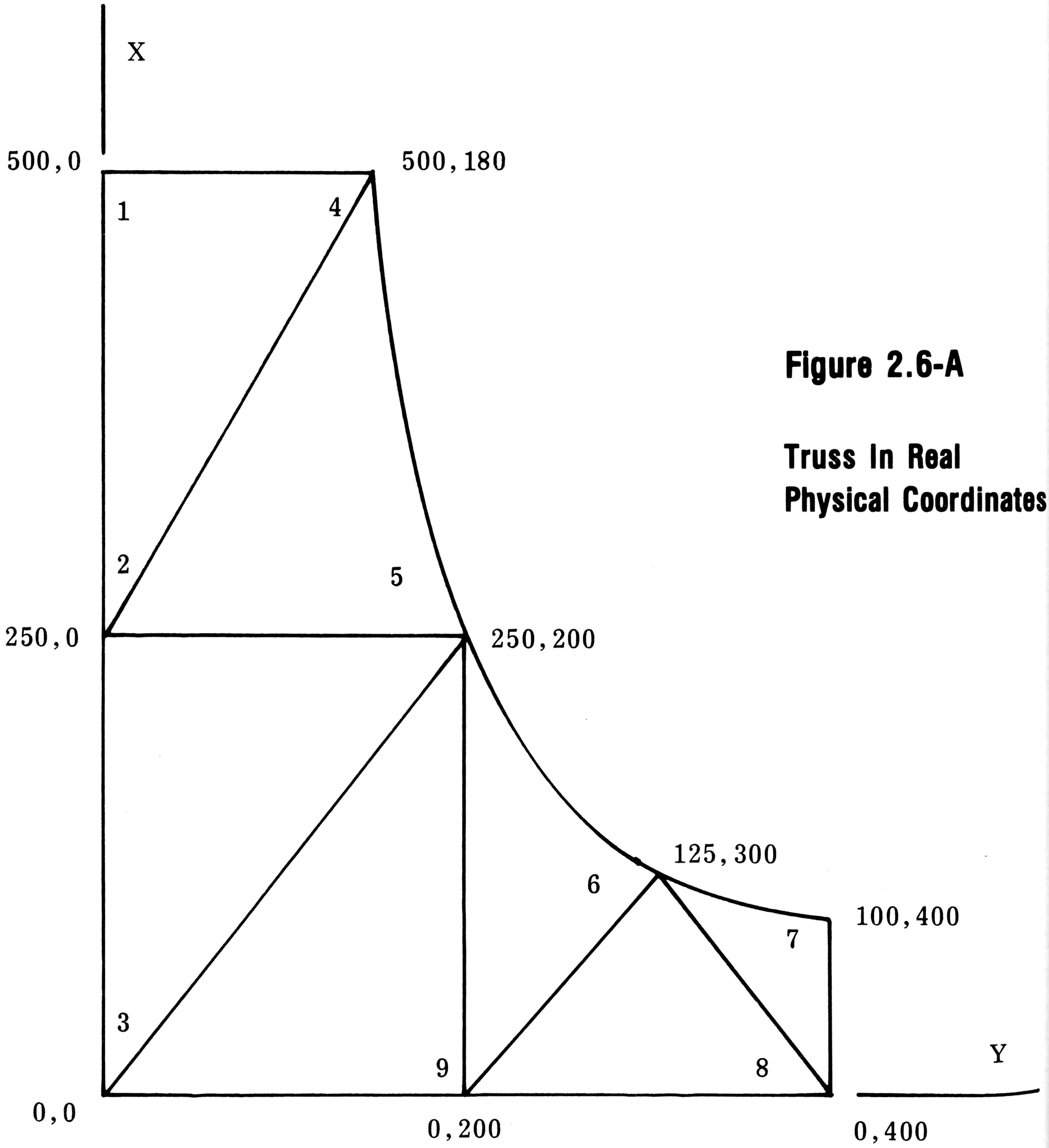

X
500,0
500,180
1
4
2
5
250,0
250,200
125,300
6
100,400
7
3
9
8
Y
0,0
0,200
0,400

Figure 2.6-A

Truss In Real Physical Coordinates

```
]RUN
CAD20
CREATES A 3D DRAWING
FROM REAL PHYSICAL COORDS
SCALES, ROTATES AND OFFSETS
PLOT COORDINATES
TYPE CONT TO CONTINUE

BREAK IN 40
]CONT
SF: .2
XO,YO (APPLE UNITS): 100,130
PX: 270
PY: 270
P,L,FILE OR END: P
POINT NO: 1
X,Y,Z COORDS: 500,0,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 4
X,Y,Z COORDS: 500,180,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 2
X,Y,Z COORDS: 250,0,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 5
X,Y,Z COORDS: 250,200,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 3
X,Y,Z COORDS: 0,0,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 9
X,Y,Z COORDS: 0,200,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 6
X,Y,Z COORDS: 125,300,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 7
X,Y,Z COORDS: 100,400,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 8
X,Y,Z COORDS: 0,400,0
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 1,4
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 1,2
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 2,4
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 4,5
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 2,5
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 2,3
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 3,9
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 5,9
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 3,5
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 5,6
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 9,6
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 8,9
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 6,8
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 6,7
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 7,8
COLOR: 7
P,L,FILE OR END: FILE
FILE NAME: CAD20.TRUSS
DRIVE NUMBER: 1
P,L,FILE OR END: END
ALL DONE
```

CAD20
CAD30

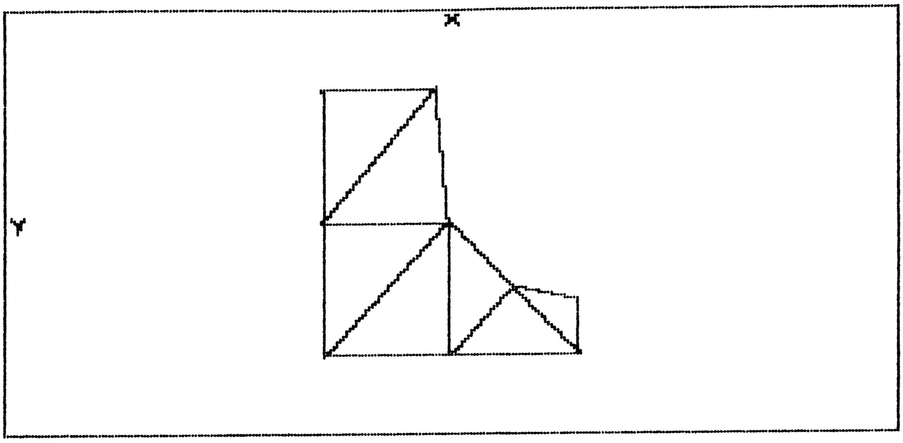


Figure 2.6-B **Truss After Transformation and Plotting**

PROGRAM *CAD30*

This program is intended to be used in conjunction with *CAD20.* *CAD20* creates a drawing in real physical units and stores it on disk. *CAD30* reads a drawing from disk, lists the data on a printer, updates the drawing, and replaces it on disk. The update feature includes the ability to move points, redefine lines, and add points and lines.

This updating capability illustrates on of the most important benefits of CAD/CAM software that was described earlier: the ability to quickly modify a drawing without having to redo the entire thing. While it may take quite a bit of time to initially create the drawing, once it has been created and stored on disk it can be easily recalled, modified and replaced on disk.

The time saved is especially evident if some sort of analysis is to be performed on the data since we not only update the graphics, but we simultaneously update the data required by the analytical software. For example, suppose we initially created a drawing, such as the truss we worked on with *CAD20,* and that we followed this with a computerized stress analysis. Then suppose that we wanted to move one node point, obtain a new drawing and a new stress analysis. All we have to do is recall the drawing from disk, redefine the position of the node, replace it on disk, then work on the new file with plot and stress programs.

In some cases, we may find that it takes *more* time to initially key in the original drawing than would be required to draw it by hand. While this may be true, it is in the subsequent operations - updating, plotting, stress analysis, etc - that the time is saved and the benefits of CAD/CAM software are realized.

PROGRAM LISTING

CAD30 contains many of the same routines that were presented in earlier programs. Only the new ones will be discussed here. The new routines are:

A - Input
B - Recall drawing from disk
C - List drawing data
D - Update drawing
E - Replot
F - Find point coordinates

In **SECTION A,** which is the input stage, a menu is written on the screen that gives the user a choice of operations: RECALL allows him to recall a drawing from the disk; CLEAR clears the screen; LIST lists the data in a drawing; UPDATE lets him change or add points and lines to a drawing; FILE places a drawing on disk; PLOT plots a drawing on the screen; END terminates program execution.

Line 115 asks which operation is to be performed and lines 118-147 transfer control to the proper subroutine where the operation is carried out. After each operation, control returns to the input section and ultimately to line 115. The only way out of the loop is by typing END.

SECTION B, which is subroutine 3000, carries out the recall of a drawing from disk. It is essentially the disk writing routine in reverse: the structure is the same as the disk writing section of *CAD20* except that the PRINT commands have been replaced by INPUT's. The data is read in exactly the same order in which it was written. After the drawing is read, line 3260 returns control to the input stage and ultimately to line 115 where a new command is asked for.

SECTION E, which is subroutine 3500, plots the drawing. Loop 3510-3580 plots the points. Loop 3590-3730 plots the lines. These operations are identical to the plotting statements in CAD20 except that we plot all points and lines at one time here. Also, we do not check for points being out of bounds or lines having nonexistent end points since these checks would have been carried out when the drawing was first created. This subroutine employs subroutines 5400, 4200 and 4000, all of which were explained in program **CAD20.**

SECTION C, which is subroutine 6000, lists the data. Line 6060 shifts output to a line printer which is assumed to be plugged in to slot 1. Point and line data is written in loops 6090-6110 and 6150-6170. Line 6180 returns output to the screen.

Subroutine 2900 clears the screen by erasing the entire screen with the HGR command and then replotting the border.

Subroutine 4800 files the drawing as in earlier progams.

SECTION D, which is subroutine 7000, updates a drawing. After typing "U", control drops to line 7000 where another menu is presented in lines 7010, 7012, and 7013. These options allow the user to change a point or a line, add a point or a line, or end the update which returns control back to line 115 in the input stage.

If "CP" is typed, meaning a point is to be changed, control drops to line 7100. Line 7110 asks for the number of the point, PU. This is the value of P(I) as input in **CAD20.** It is not the value of I. For example, when the drawing was created, if the second point that was input was assigned the number 12345, then P(2) = 12345. If we want to change the location of this point, we would specify 12345 for the value of PU in line 7110. Control then drops to subroutine 8000 which finds the corresponding value of I in the P(I) array. This search through the pointer array was explained in program **CAD20.** The location of point PU in the P(I) array is assigned the temporary value, A.

Lines 7120 and 7130 ask for new coordinates and a new color. Line 7135 drops control to subroutine 7700 where the screen is erased and a new drawing is plotted. Control then returns to 7010 where a new update command is asked for.

If a line is to be changed, control drops to 7500. The procedure is the same except that we define a line by its location in the LB(I), LE(I) and LC(I) arrays. This is, we must specify I. To facilitate this, it is best to first get a listing of the drawing before doing the updating.

Points are added starting at line 7550. The procedure is the same as changing a point except that the position of the point in the arrays is set to NP + 1. Lines are added in a similar manner starting at line 7575.

The structure of this program is imporatnt. Note that the program uses a menu-command-operation-return to menu sequence.

When a particular command is specified, control drops to whatever subroutine is responsible for carrying out that particular operation.

This structure makes the program easy to follow and fix. It also allows us to add new operations with little difficulty. For example, suppose we want to add physical properties to the lines in a drawing. We may, for example, want the lines to represent connectors in an electric circuit, in which case we would want to attach electrical properties to the lines. Or they may represent beams in a structure, in which case we might want to give them weights and dimensions. This could be done neatly by adding an option to the menu which we might call "PROPERTY". If the user types "PR", control would drop to a subroutine where the user would key in the member number (I) and the required physical properties. These would be stored in arrays. We could even assign physical properties to the nodes in the same manner.

This raises the question of whether a program like **CAD30** should be specialized to a particular class of problems or whether it should be general purpose. If we want to make it general purpose, we would assign a number of arrays to each line and node for the storage of physical properties. The user could then define these to be anything he wanted. For example, suppose we had five arrays for lines: V(1), V(2)...V(5). V(1) could represent cross sectional area of a beam; V(2) might be specific weight of the material; V(3) could represent Young's Modulus, etc. These data would be stored on disk as part of the drawing file. Any applications program that read the drawing from disk would, of course, read these data as well.

```
10  PRINT "CAD30"
11  PRINT "RECALLS A DRAWING FROM DISK"
12  PRINT "LISTS DATA, UPDATES, AND"
13  PRINT "REPLACES DRAWING ON DISK"
30  PRINT "TYPE CONT TO CONTINUE"
40  STOP
50  HGR
60  HCOLOR= 7
70  GOSUB 9990
80  DIM P(50),XP(50),YP(50),ZP(50),PC(50)
90  DIM LB(40),LE(40),LC(40)
96  REM
97  REM
98  REM
99  REM
100  REM  INPUT
105  PRINT "R=RECALL"; TAB( 12);"C=CLEAR"; TAB( 24);"L=LIST"
110  PRINT "U=UPDATE"; TAB( 12);"F=FILE"; TAB( 24);"E=END"
112  PRINT "P=PLOT"
```

A - Control

```
115  INPUT "COMMAND: ";CK$
118  IF CK$ = "E" GOTO 9998
120  IF CK$ = "R" THEN  GOSUB 3000
125  IF CK$ = "C" THEN  GOSUB 2900
130  IF CK$ = "L" THEN  GOSUB 6000
135  IF CK$ = "U" THEN  GOSUB 7000
140  IF CK$ = "F" THEN  GOSUB 4800
145  IF CK$ = "P" THEN  GOSUB 3500
150  GOTO 105
2896  REM
2897  REM
2899  REM
2900  REM  CLEAR SCREEN
2905  PRINT "CLEAR"
2910  HGR
2915  GOSUB 9990
2929  RETURN
2996  REM
2997  REM
2998  REM
2999  REM
3000  REM  RECALL DRAWING FROM DISK
3005  PRINT "RECALL"
3010  INPUT "FILE NAME: ";F$
3020  INPUT "DRIVE NUMBER: ";DR
3030 D$ = "": REM  CTRL-D
3040  PRINT D$;"OPEN";F$;",D";DR
3050  PRINT D$;"READ";F$
3060  INPUT SF
3070  INPUT X0
3080  INPUT Y0
3090  INPUT PX
3100  INPUT PY
3110  INPUT NP
3120  FOR I = 1 TO NP
3130  INPUT P(I)
3140  INPUT XP(I)
3150  INPUT YP(I)
3160  INPUT ZP(I)
3170  INPUT PC(I)
3180  NEXT I
3190  INPUT NL
3200  FOR I = 1 TO NL
3210  INPUT LB(I)
3220  INPUT LE(I)
3230  INPUT LC(I)
3240  NEXT I
3250  PRINT D$;"CLOSE"
3260  RETURN
3496  REM
3497  REM
```

B - Recalls drawing from disk

```
3498  REM
3499  REM
3500  REM  PLOT
3505  PRINT "PLOT"
3510  FOR I = 1 TO NP
3520 XR = XP(I)
3530 YR = YP(I)
3540  GOSUB 5400
3550  GOSUB 4000
3560  HCOLOR= PC(I)
3570  HPLOT XT,YT
3580  NEXT I
3590  FOR I = 1 TO NL
3595 LN = I
3600  GOSUB 4200
3610 XR = XP(A)
3620 YR = YP(A)
3630  GOSUB 5400
3640 AA = XT
3650 BB = YT
3660 XR = XP(B)
3670 YR = YP(B)
3680  GOSUB 5400
3690 CC = XT
3700 DD = YT
3710  HCOLOR= LC(I)
3720  HPLOT AA,BB TO CC,DD
3730  NEXT I
3740  RETURN
3996  REM
3997  REM
3998  REM
3999  REM
4000  REM  OUT OF BOUNDS CHECK
4005 OB = 0
4010  IF XT < 0 GOTO 4100
4020  IF XT > 279 GOTO 4110
4030  IF YT < 0 GOTO 4120
4040  IF YT > 159 GOTO 4130
4050  GOTO 4140
4100  PRINT "XT<0-TRY AGAIN"
4105  GOTO 4135
4110  PRINT "XT>279-TRY AGAIN"
4115  GOTO 4135
4120  PRINT "YT<0-TRY AGAIN"
4125  GOTO 4135
4130  PRINT "YT>0-TRY AGAIN"
4135 OB = 1
4140  RETURN
4196  REM
4197  REM
```

```
4198  REM
4199  REM
4200  REM  FIND POINTS
4205 MP = 0
4210  FOR K = 1 TO NP
4220  IF LB(LN) = P(K) THEN A = K
4230  IF LB(LN) = P(K) GOTO 4270
4240  NEXT K
4250  PRINT "CAN'T FIND POINT ";LB(LN)
4260 MP = 1
4265  GOTO 4330
4270  FOR K = 1 TO NP
4280  IF LE(LN) = P(K) THEN B = K
4290  IF LE(LN) = P(K) GOTO 4330
4300  NEXT K
4310  PRINT "CAN'T FIND POINT ";LE(LN)
4315 MP = 1
4330  RETURN
4796  REM
4797  REM
4798  REM
4799  REM
4800  REM  FILE DRAWING ON DISK
4805  PRINT "FILE"
4860  INPUT "FILE NAME: ";F$
4870  INPUT "DRIVE NUMBER: ";DR
4880 D$ = "": REM  CTRL-D
4890  PRINT D$;"OPEN";F$;",D";DR
4900  PRINT D$;"WRITE";F$
4905  PRINT SF
4906  PRINT XO
4907  PRINT YO
4908  PRINT PX
4909  PRINT PY
4910  PRINT NP
4920  FOR I = 1 TO NP
4930  PRINT P(I)
4940  PRINT XP(I)
4950  PRINT YP(I)
4960  PRINT ZP(I)
4970  PRINT PC(I)
4980  NEXT I
4990  PRINT NL
5000  FOR I = 1 TO NL
5010  PRINT LB(I)
5020  PRINT LE(I)
5030  PRINT LC(I)
5040  NEXT I
5045  PRINT D$;"CLOSE"
5046  HGR
5047  GOSUB 9990
5050  RETURN
5396  REM
5397  REM
```

```
5398  REM
5399  REM
5400  REM  SCALE, ROTATE AND OFFSET PLOT COORDINATES
5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
5996  REM
5997  REM
5998  REM
5999  REM
```

C - Lists drawing data

```
6000  REM  LIST DRAWING DATA
6005  PRINT "LIST"
6060  PRINT D$;"PR#1"
6070  PRINT "POINTS"
6080  PRINT
6090  FOR I = 1 TO NP
6100  PRINT I; TAB( 8);P(I); TAB( 16);XP(I); TAB( 24);YP(I); TAB( 32);ZP(I
     ); TAB( 40);PC(I)
6110  NEXT I
6120  PRINT
6130  PRINT
6140  PRINT "LINES"
6145  PRINT
6150  FOR I = 1 TO NL
6160  PRINT I; TAB( 8);LB(I); TAB( 16);LE(I); TAB( 24);LC(I)
6170  NEXT I
6180  PRINT D$;"PR#0"
6200  RETURN
6296  REM
6297  REM
6298  REM
6299  REM
```

D - Updates a drawing

```
7000  REM  UPDATE DRAWING
7010  PRINT "CP=CHANGE POINT"; TAB( 20);"CL=CHANGE LINE"
7012  PRINT "AP=ADD POINT"; TAB( 20);"AL=ADD LINE"
7013  PRINT "E=END"
7014  INPUT "COMMAND: ";CK$
7015  IF CK$ = "E" GOTO 7600
7016  IF CK$ = "CP" GOTO 7100
7020  IF CK$ = "CL" GOTO 7500
7030  IF CK$ = "AP" GOTO 7550
7040  IF CK$ = "AL" GOTO 7575
7050  GOTO 7010
7100  REM  CHANGE A POINT
7110  INPUT "POINT NUMBER: ";PU
7112  GOSUB 8000
7115  IF MP = 1 GOTO 7600
7120  INPUT "NEW COORDS: ";XP(A),YP(A),ZP(A)
7130  INPUT "NEW COLOR: ";PC(A)
7135  GOSUB 7700
7140  GOTO 7010
7500  REM  CHANGE A LINE
7510  INPUT "LINE NUMBER: ";I
7520  INPUT "NEW NODES: ";LB(I),LE(I)
```

```
7530  INPUT "NEW COLOR: ";LC(I)
7535  GOSUB 7700
7540  GOTO 7010
7550  REM  ADD A POINT
7554 NP = NP + 1
7555  INPUT "POINT NUMBER: ";P(NP)
7560  INPUT "COORDS: ";XP(NP),YP(NP),ZP(NP)
7565  INPUT "COLOR: ";PC(NP)
7570  GOSUB 7700
7571  GOTO 7010
7575  REM  ADD A LINE
7580 NL = NL + 1
7585  INPUT "NODES: ";LB(NL),LE(NL)
7590  INPUT "COLOR: ";LC(NL)
7595  GOSUB 7700
7596  GOTO 7010
7600  RETURN
7696  REM
7697  REM
7698  REM
7699  REM
```

E - Replots a drawing

```
7700  REM  RE-PLOT
7710  HGR
7720  GOSUB 9990
7730  GOSUB 3500
7740  RETURN
```

F - Finds point coordinates

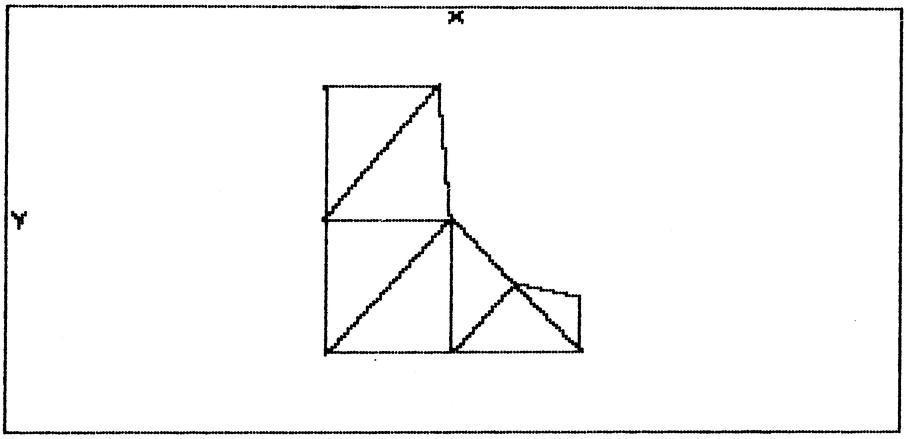
```
8000  REM  PINF POINT COORDINATES
8005 MP = 0
8010  FOR K = 1 TO NP
8020  IF PU = P(K) THEN A = K
8030  IF PU = P(K) GOTO 8050
8040  NEXT K
8045  PRINT "CAN'T FIND POINT ";PU
8046 MP = 1
8050  RETURN
9986  REM
9987  REM
9988  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 136,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

SAMPLE RUN

```
CAD30
RECALLS A DRAWING FROM DISK
LISTS DATA, UPDATES, AND
REPLACES DRAWING ON DISK
TYPE CONT TO CONTINUE
```

CAD30

```
BREAK IN 40
]CONT
R=RECALL    C=CLEAR     L=LIST
U=UPDATE    F=FILE      E=END
P=PLOT
COMMAND: R
RECALL
FILE NAME: CAD20.TRUSS
DRIVE NUMBER: 1
R=RECALL    C=CLEAR     L=LIST
U=UPDATE    F=FILE      E=END
P=PLOT
COMMAND: P
PLOT
```

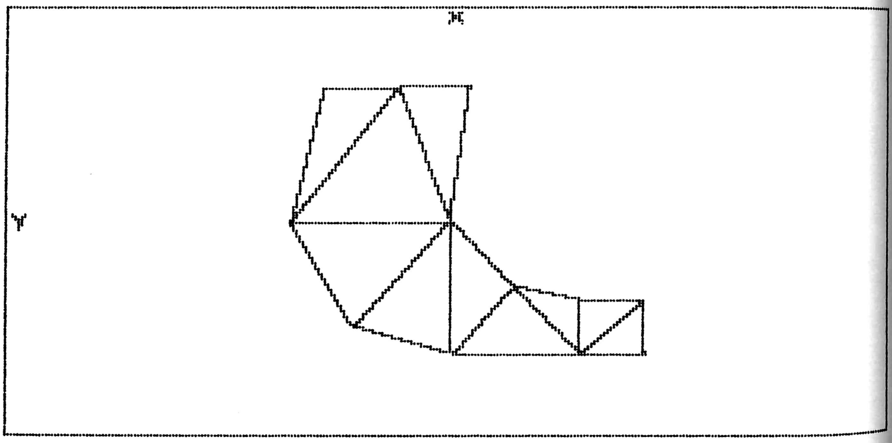


```
R=RECALL    C=CLEAR     L=LIST
U=UPDATE    F=FILE      E=END
P=PLOT
COMMAND: L
LIST


POINTS

1       1       500     0       0       7
2       4       500     180     0       7
3       2       250     0       0       7
4       5       250     200     0       7
5       3       0       0       0       7
6       9       0       200     0       7
7       6       125     300     0       7
8       7       100     400     0       7
9       8       0       400     0       7
```

```
LINES

1         1         4         7
2         1         2         7
3         2         4         7
4         4         5         7
5         2         5         7
6         2         3         7
7         3         9         7
8         5         9         7
9         3         5         7
10        5         6         7
11        9         6         7
12        8         9         7
13        6         8         7
14        6         7         7
15        7         8         7


COMMAND: U
CP=CHANGE POINT      CL=CHANGE LINE
AP=ADD POINT         AL=ADD LINE
E=END
COMMAND: AP
POINT NUMBER: 10
COORDS: 100,500,0
COLOR: 7
PLOT
CP=CHANGE POINT      CL=CHANGE LINE
AP=ADD POINT         AL=ADD LINE
E=END
COMMAND: AP
POINT NUMBER: 11
COORDS: 0,500,0
COLOR: 7
PLOT
CP=CHANGE POINT      CL=CHANGE LINE
AP=ADD POINT         AL=ADD LINE
E=END
COMMAND: AL
NODES: 7,10
COLOR: 7
PLOT
CP=CHANGE POINT      CL=CHANGE LINE
AP=ADD POINT         AL=ADD LINE
E=END
COMMAND: AL
NODES: 8,11
COLOR: 7
PLOT
CP=CHANGE POINT      CL=CHANGE LINE
AP=ADD POINT         AL=ADD LINE
E=END
```

```
COMMAND: AL
NODES: 10,11
COLOR: 7
PLOT
CP=CHANGE POINT     CL=CHANGE LINE
AP=ADD POINT        AL=ADD LINE
E=END
COMMAND: AL
NODES: 8,10
COLOR: 7
PLOT
CP=CHANGE POINT     CL=CHANGE LINE
AP=ADD POINT        AL=ADD LINE
E=END
COMMAND: CP
POINT NUMBER: 2
NEW COORDS: 250,-50,0
NEW COLOR: 7
PLOT
CP=CHANGE POINT     CL=CHANGE LINE
AP=ADD POINT        AL=ADD LINE
E=END
COMMAND: CP
POINT NUMBER: 3
NEW COORDS: 20,20,0
NEW COLOR: 7
PLOT
CP=CHANGE POINT     CL=CHANGE LINE
AP=ADD POINT        AL=ADD LINE
E=END
COMMAND: CP
POINT NUMBER: 3
NEW COORDS: 50,50,0
NEW COLOR: 7
PLOT
CP=CHANGE POINT     CL=CHANGE LINE
AP=ADD POINT        AL=ADD LINE
E=END
COMMAND: AP
POINT NUMBER: 12
COORDS: 500,230,0
COLOR: 7
PLOT
CP=CHANGE POINT     CL=CHANGE LINE
AP=ADD POINT        AL=ADD LINE
E=END
COMMAND: AL
NODES: 4,12
COLOR: 7
PLOT
CP=CHANGE POINT     CL=CHANGE LINE
AP=ADD POINT        AL=ADD LINE
E=END
```

```
COMMAND: AL
NODES: 5,12
COLOR: 7
PLOT
CP=CHANGE POINT    CL=CHANGE LINE
AP=ADD POINT       AL=ADD LINE
E=END
COMMAND: CP
POINT NUMBER: 4
NEW COORDS: 500,120,0
NEW COLOR: 7
PLOT
CP=CHANGE POINT    CL=CHANGE LINE
AP=ADD POINT       AL=ADD LINE
E=END
COMMAND: E
ALL DONE
```



PROGRAM CAD40

This program can be used to recall and display several drawings simultaneously. The drawings can also be moved around the screen.

CAD40

The main sections of the program are similar to those in **CAD30** with a few exceptions. First, when we erase a drawing, we erase only one drawing, leaving the others on the screen. Therefore, we cannot use the HGR command, which erases the entire screen. Subroutine 6000 shows how we erase only one drawing at a time. We set an "erase switch", ES, equal to 1 and transfer control to subroutine 3500 which does the plotting. That subroutine has been modified to set the plotting color to black (HCOLOR = 0) if ES is equal to 1. In that way, we redraw a drawing in black which erases it.

The second modification is in the addition of subroutine 7000 which moves a drawing. First it is erased by a call to subroutine 6000. Next, new offset values xo,yo are defined and the plotting subroutine 3500 is called. This redraws the drawing in its new position.

Note that this program does not include provision for storing the merged drawings. This would, in principle, be an easy thing to do. All we would have to do is add the proper command to the menu and insert a disk writing routine such as was used in previous programs. The difficulty with this approach is that each of the drawings on the screen may have a different set of scaling, offset and rotation parameters. If a disk writing subroutine is inserted in this program, it should write each drawing to the file as a separate drawing, each with its own parameters. The programs that read the file would then have to know how many drawing were on the file. This could be the first item on the file.

```
10  PRINT "CAD40"
11  PRINT "DISPLAYS SEVERAL DRAWINGS"
12  PRINT "SIMULTANEOUSLY"
30  PRINT "TYPE CONT TO CONTINUE"
40  STOP
50  HGR
60  HCOLOR= 7
70  GOSUB 9990
80  DIM P(50),XP(50),YP(50),ZP(50),PC(50)
90  DIM LB(25),LE(25),LC(25)
91  REM
92  REM
93  REM
94  REM
95  REM  INPUT
100  PRINT "R=RECALL"; TAB( 12);"C=CLEAR"
105  PRINT "M=MOVE"; TAB( 12);"E=END"
110  INPUT "COMMAND: ";CK$
```

```
  120  IF CK$ = "E" GOTO 9998
  130  IF CK$ = "R" THEN  GOSUB 3000
  140  IF CK$ = "C" THEN  GOSUB 6000
  150  IF CK$ = "M" THEN  GOSUB 7000
  160  GOTO 100
  2996  REM
  2997  REM
  2998  REM
  2999  REM
  3000  REM  RECALL DRAWING FROM DISK
  3010  INPUT "FILE NAME: ";F$
  3020  INPUT "DRIVE NUMBER: ";DR
  3030 D$ = "": REM  CTRL-D
  3040  PRINT D$;"OPEN ";F$;",D";DR
  3050  PRINT D$;"READ ";F$
  3060  INPUT SF
  3070  INPUT XO
  3080  INPUT YO
  3090  INPUT PX
  3100  INPUT PY
  3110  INPUT NP
  3120  FOR I = 1 TO NP
  3130  INPUT P(I)
  3140  INPUT XP(I)
  3150  INPUT YP(I)
  3160  INPUT ZP(I)
  3170  INPUT PC(I)
  3180  NEXT I
  3190  INPUT NL
  3200  FOR I = 1 TO NL
  3210  INPUT LB(I)
  3220  INPUT LE(I)
  3230  INPUT LC(I)
  3240  NEXT I
  3250  PRINT D$;"CLOSE"
  3255  GOSUB 3500
➤ 3256  PRINT "XO= ";XO; TAB( 10);"YO= ";YO
  3260  RETURN
  3496  REM
  3497  REM
  3498  REM
  3499  REM
  3500  REM  PLOT
  3510  FOR I = 1 TO NP
  3520 XR = XP(I)
  3530 YR = YP(I)
  3540  GOSUB 5400
  3541  HCOLOR= 0
➤ 3542  IF ES = 1 THEN  GOTO 3570
  3560  HCOLOR= PC(I)
  3570  HPLOT XT,YT
  3580  NEXT I
  3590  FOR I = 1 TO NL
  3595 LN = I
  3600  GOSUB 4200
  3610 XR = XP(A)
  3620 YR = YP(A)
  3630  GOSUB 5400
  3640 AA = XT
  3650 BB = YT
  3660 XR = XP(B)
  3670 YR = YP(B)
  3680  GOSUB 5400
  3690 CC = XT
  3700 DD = YT
  3701  HCOLOR= 0
➤ 3702  IF ES = 1 THEN  GOTO 3720
  3710  HCOLOR= LC(I)
  3720  HPLOT AA,BB TO CC,DD
  3730  NEXT I
  3740  RETURN
  4196  REM
  4197  REM
  4198  REM
  4199  REM
  4200  REM  FIND POINTS
  4210  FOR K = 1 TO NP
  4220  IF LB(LN) = P(K) THEN A = K
  4230  IF LB(LN) = P(K) GOTO 4270
  4240  NEXT K
  4250  PRINT "CAN'T FIND POINT ";LB(LN)
  4260  GOTO 3110
  4270  FOR K = 1 TO NP
  4280  IF LE(LN) = P(K) THEN B = K
  4290  IF LE(LN) = P(K) GOTO 4330
  4300  NEXT K
  4310  PRINT "CAN'T FIND POINT ";LE(LN)
  4320  GOTO 3110
  4330  RETURN
  5396  REM
  5397  REM
  5398  REM
  5399  REM
  5400  REM  SCALE,ROTATE AND OFFSET PLOT COORDS
  5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
  5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
  5430  RETURN
  5996  REM
  5997  REM
  5998  REM
  5999  REM
  6000  REM  ERASE A DRAWING
  6010 ES = 1
  6020  GOSUB 3500
  6030 ES = 0
  6040  RETURN
  6996  REM
  6997  REM
```

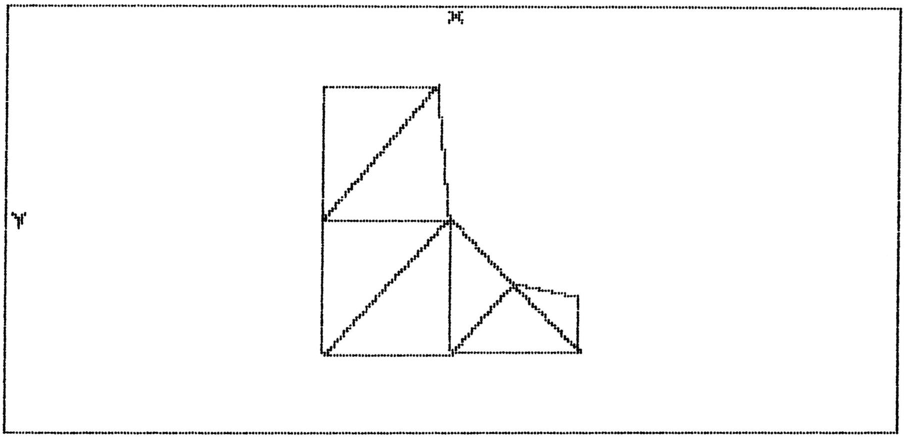
```
6998  REM
6999  REM
7000  REM  MOVE A DRAWING
7010  INPUT "NEW X0,Y0: ";J1,J2
7020  GOSUB 6000
7025 X0 = J1
7026 Y0 = J2
7030  GOSUB 3500
7035  PRINT "X0= ";X0; TAB( 10);"Y0= ";Y0
7040  RETURN
9986  REM
9987  REM
9988  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

SAMPLE RUN

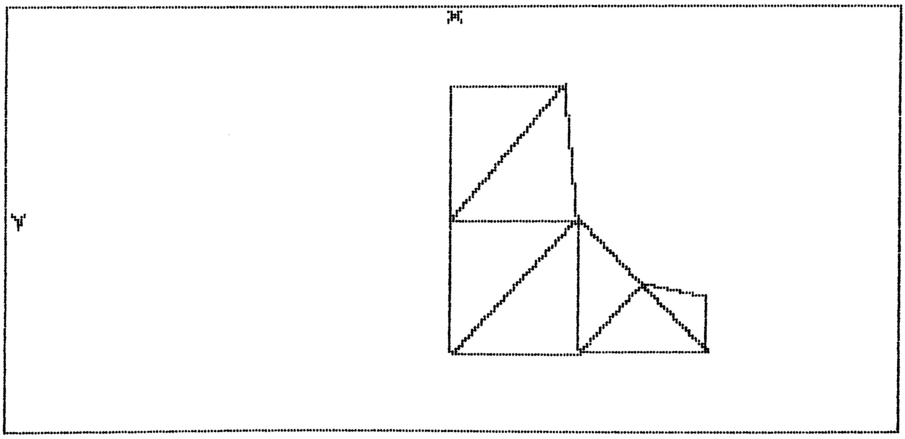
In the sample run that follows the program listing, we recall two drawings from disk, move one of them, and display the two simultaneously.

CAD40

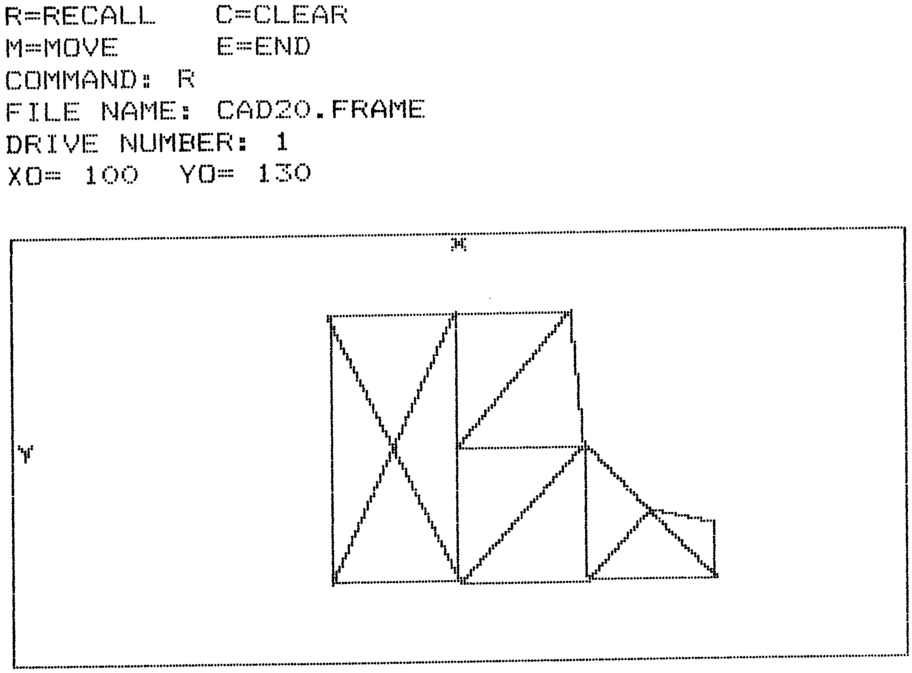
```
CAD40
DISPLAYS SEVERAL DRAWINGS
SIMULTANEOUSLY
TYPE CONT TO CONTINUE

BREAK IN 40
]CONT
R=RECALL    C=CLEAR
M=MOVE      E=END
COMMAND: R
FILE NAME: CAD20.TRUSS
DRIVE NUMBER: 1
XO= 100   YO= 130
```



```
R=RECALL    C=CLEAR
M=MOVE      E=END
COMMAND: M
NEW XO,YO: 140,130
XO= 140   YO= 130
```



CAD40
CAD50



```
R=RECALL    C=CLEAR
M=MOVE      E=END
COMMAND: R
FILE NAME: CAD20.FRAME
DRIVE NUMBER: 1
XO= 100  YO= 130
```

PROGRAM *CAD50*

As mentioned in the discussion of *CAD30,* it is a relatively simple matter to add physical properties to either the points or lines in a drawing. In this program we add properties to lines. The procedure is to first create the geometry of a drawing using either *CAD20* or *CAD30,* then run *CAD50* and add the properties. *CAD50* operates much like *CAD30* in that it can recall drawings from disk and update them. In fact, it is almost identical to *CAD30* except that we have removed the section that updates points and lines and added a section that adds physical properties to existing lines.

You may ask why this operation couldn't have been included in *CAD30?* The reason is that the memory of the Apple II is not large enough to hold such a large program along with a reasonable amount of point and line data. When we try to add the physical property routine to *CAD30,* it infringes upon the portion of memory that holds the Page 1 hi-res graphics information. As a result, part of the screen image is destroyed. This is a good example of the philosophy referred to earlier in this chapter - the limited active memories of microcomputers require that complex operations be carried out in small steps by separate programs with the intermediate data being stored on disk.

It should be mentioned, that the programs in this book have not been optimized for minimum memory requirements. This is because we have deliberately coded them in an open style to enhance readability. With a little effort, you should be able to reduce them in size considerably and combine many of the operations into one large program.

PROGRAM LISTING

Referring to the program listing, an "ADD PROPERTIES" command has been added to the menu in line 112. Line 148 shifts control to subroutine 8100. This subroutine has been specialized to assign material properties to the lines of the drawing. It could be modified to assign any type of properties, or could even be made general. It presents a menu of materials: aluminum, titanium, steel, and magnesium. The material code is entered in line 8115. The number of the drawing line which is to be assigned this material is input in line 8120. Control then drops to subroutine 8200 where the variable DE(I) is set equal to the material density (lbs/in^2). The user has to type in only the material code - AL, ST, TI or MA - the program searches for the associated material properties.

We could have added as many properties as we wanted in addition to the density. For example, we could have set up arrays for Young's Modulus, thermal conductivity, Poisson's ratio, etc. This procedure eliminates the chore of typing in these properties every time we wish to define a line. All we have to do is specify the material and the software finds the associated physical properties and attaches them to the line. A subroutine like 8200 is called a "library" or "catalog" since it contains specifications corresponding to many different type of materials or parts. Note that the routine which prints the drawing file has been modified to print the material properties.

We can use *CAD50* more than once on the same drawing. For example, when we used *CAD20* we developed a drawing called CAD20.TRUSS. We can recall that drawing with *CAD50* , add material properties, and replace it on disk under the same name, or a different one such as CAD50.TRUSS. This will be the same as the original CAD20.TRUSS but the file will contain material properties for the lines. We could recall CAD50.TRUSS later and change the material properties if we wish.

CAD50

```
    10  PRINT "CAD50"
    11  PRINT "RECALLS A DRAWING"
    13  PRINT "LISTS DATA"
    14  PRINT "ADDS PROPERTIES TO LINES"
    15  PRINT "REPLACES ON DISK"
    30  PRINT "TYPE CONT TO CONTINUE"
    40  STOP
    50  HGR
    60  HCOLOR= 7
    70  GOSUB 9990
    80  DIM P(50),XP(50),YP(50),ZP(50),PC(50)
    90  DIM LB(25),LE(25),LC(25)
 ➤  95  DIM M$(25),AR(25),DE(25)
    96  REM
    97  REM
    98  REM
    99  REM
    100  REM  INPUT
    105  PRINT "R=RECALL"; TAB( 12);"C=CLEAR"; TAB( 24);"L=LIST"
    110  PRINT "F=FILE"; TAB( 12);"E=END"
 ➤  112  PRINT "P=PLOT"; TAB( 12);"A=ADD PROPERTIES"
    115  INPUT "COMMAND: ";CK$
    118  IF CK$ = "E" GOTO 9998
    120  IF CK$ = "R" THEN  GOSUB 3000
    125  IF CK$ = "C" THEN  GOSUB 2900
    130  IF CK$ = "L" THEN  GOSUB 6000
    140  IF CK$ = "F" THEN  GOSUB 4800
    145  IF CK$ = "P" THEN  GOSUB 3500
 ➤  148  IF CK$ = "A" THEN  GOSUB 8100
    150  GOTO 105
    2896  REM
    2897  REM
    2899  REM
    2900  REM  CLEAR SCREEN
    2905  PRINT "CLEAR"
    2910  HGR
    2915  GOSUB 9990
    2929  RETURN
    2996  REM
    2997  REM
    2998  REM
    2999  REM
    3000  REM  RECALL DRAWING FROM DISK
    3005  PRINT "RECALL"
    3010  INPUT "FILE NAME: ";F$
    3020  INPUT "DRIVE NUMBER: ";DR
    3030 D$ = "": REM  CTRL-D
    3040  PRINT D$;"OPEN";F$;",D";DR
    3050  PRINT D$;"READ";F$
    3060  INPUT SF
    3070  INPUT XO
    3080  INPUT YO
    3090  INPUT PX
```

```
3100  INPUT PY
3110  INPUT NP
3120  FOR I = 1 TO NP
3130  INPUT P(I)
3140  INPUT XP(I)
3150  INPUT YP(I)
3160  INPUT ZP(I)
3170  INPUT PC(I)
3180  NEXT I
3190  INPUT NL
3200  FOR I = 1 TO NL
3210  INPUT LB(I)
3220  INPUT LE(I)
3230  INPUT LC(I)
3240  NEXT I
3250  PRINT D$;"CLOSE"
3260  RETURN
3496  REM
3497  REM
3498  REM
3499  REM
3500  REM  PLOT
3505  PRINT "PLOT"
3510  FOR I = 1 TO NP
3520 XR = XP(I)
3530 YR = YP(I)
3540  GOSUB 5400
3550  GOSUB 4000
3560  HCOLOR= PC(I)
3570  HPLOT XT,YT
3580  NEXT I
3590  FOR I = 1 TO NL
3595 LN = I
3600  GOSUB 4200
3610 XR = XP(A)
3620 YR = YP(A)
3630  GOSUB 5400
3640 AA = XT
3650 BB = YT
3660 XR = XP(B)
3670 YR = YP(B)
3680  GOSUB 5400
3690 CC = XT
3700 DD = YT
3710  HCOLOR= LC(I)
3720  HPLOT AA,BB TO CC,DD
3730  NEXT I
3740  RETURN
3996  REM
3997  REM
3998  REM
3999  REM
4000  REM  OUT OF BOUNDS CHECK
```

```
4005 OB = 0
4010  IF XT < 0 GOTO 4100
4020  IF XT > 279 GOTO 4110
4030  IF YT < 0 GOTO 4120
4040  IF YT > 159 GOTO 4130
4050  GOTO 4140
4100  PRINT "XT<0-TRY AGAIN"
4105  GOTO 4135
4110  PRINT "XT>279-TRY AGAIN"
4115  GOTO 4135
4120  PRINT "YT<0-TRY AGAIN"
4125  GOTO 4135
4130  PRINT "YT>0-TRY AGAIN"
4135 OB = 1
4140  RETURN
4196  REM
4197  REM
4198  REM
4199  REM
4200  REM  FIND POINTS
4205 MP = 0
4210  FOR K = 1 TO NP
4220  IF LB(LN) = P(K) THEN A = K
4230  IF LB(LN) = P(K) GOTO 4270
4240  NEXT K
4250  PRINT "CAN'T FIND POINT ";LB(LN)
4260 MP = 1
4265  GOTO 4330
4270  FOR K = 1 TO NP
4280  IF LE(LN) = P(K) THEN B = K
4290  IF LE(LN) = P(K) GOTO 4330
4300  NEXT K
4310  PRINT "CAN'T FIND POINT ";LE(LN)
4315 MP = 1
4330  RETURN
4796  REM
4797  REM
4798  REM
4799  REM
4800  REM  FILE DRAWING ON DISK
4805  PRINT "FILE"
4860  INPUT "FILE NAME: ";F$
4870  INPUT "DRIVE NUMBER: ";DR
4880 D$ = "": REM  CTRL-D
4890  PRINT D$;"OPEN";F$;",D";DR
4900  PRINT D$;"WRITE";F$
4905  PRINT SF
4906  PRINT XO
4907  PRINT YO
4908  PRINT PX
4909  PRINT PY
4910  PRINT NP
4920  FOR I = 1 TO NP
```

```
  4930  PRINT P(I)
  4940  PRINT XP(I)
  4950  PRINT YP(I)
  4960  PRINT ZP(I)
  4970  PRINT PC(I)
  4980  NEXT I
  4990  PRINT NL
  5000  FOR I = 1 TO NL
  5010  PRINT LB(I)
  5020  PRINT LE(I)
  5030  PRINT LC(I)
➤ 5032  PRINT M$(I)
➤ 5033  PRINT AR(I)
➤ 5034  PRINT DE(I)
  5040  NEXT I
  5045  PRINT D$;"CLOSE"
  5046  HGR
  5047  GOSUB 9990
  5050  RETURN
  5396  REM
  5397  REM
  5398  REM
  5399  REM
  5400  REM  SCALE, ROTATE AND OFFSET PLOT COORDINATES
  5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
  5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
  5430  RETURN
  5996  REM
  5997  REM
  5998  REM
  5999  REM
  6000  REM  LIST DRAWING DATA
  6005  PRINT "LIST"
  6060  PRINT D$;"PR#1"
  6070  PRINT "POINTS"
  6080  PRINT
  6090  FOR I = 1 TO NP
  6100  PRINT I; TAB( 8);P(I); TAB( 16);XP(I); TAB( 24);YP(I); TAB( 32);ZP(I
     ); TAB( 40);PC(I)
  6110  NEXT I
  6120  PRINT
  6130  PRINT
  6140  PRINT "LINES"
  6145  PRINT
  6150  FOR I = 1 TO NL
  6160  PRINT I; TAB( 8);LB(I); TAB( 16);LE(I); TAB( 24);LC(I)
➤ 6165  PRINT M$(I); TAB( 8);DE(I); TAB( 16);AR(I)
  6166  PRINT
  6170  NEXT I
  6180  PRINT D$;"PR#0"
  6200  RETURN
  6296  REM
  6297  REM
  6298  REM
```

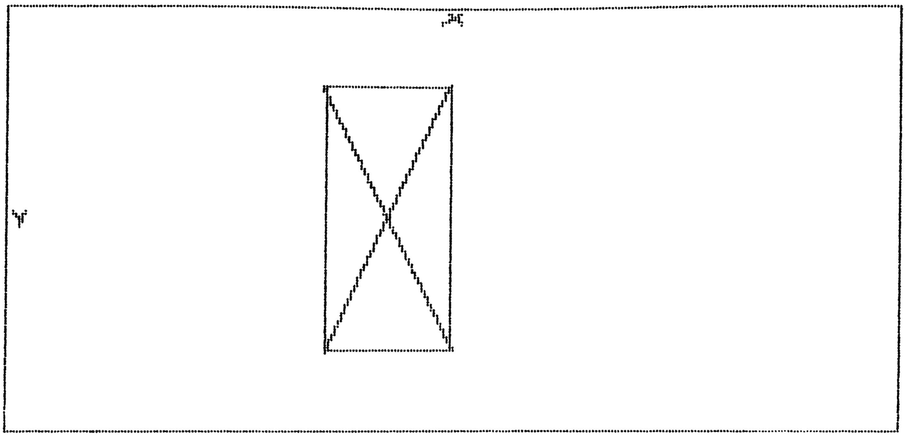
```
6299  REM
7696  REM
7697  REM
7698  REM
7699  REM
7700  REM  RE-PLOT
7710  HGR
7720  GOSUB 9990
7730  GOSUB 3500
7740  RETURN
8000  REM  FIND POINT COORDINATES
8005 MP = 0
8010  FOR K = 1 TO NP
8020  IF PU = P(K) THEN A = K
8030  IF PU = P(K) GOTO 8050
8040  NEXT K
8045  PRINT "CAN'T FIND POINT ";PU
8046 MP = 1
8050  RETURN
8096  REM
8097  REM
8098  REM
8099  REM
8100  REM  ADD PROPERTIES TO LINES
8105  PRINT "ADD PROPERTIES"
8112  PRINT "AL=ALUMINUM"; TAB( 15);"TI=TITANIUM"; TAB( 30);"ST=STEEL"
8113  PRINT "MA=MAGNESIUM"; TAB( 15);"E=END"
8115  INPUT "MATERIAL: ";A$
8116  IF A$ = "E" GOTO 8150
8120  INPUT "LINE: ";I
8125 M$(I) = A$
8128  GOSUB 8200
8130  INPUT "AREA: ";AR(I)
8140  GOTO 8112
8150  RETURN
8196  REM
8197  REM
8198  REM
8199  REM
8200  REM  FIND MATERIAL DENSITY
8210  IF M$(I) = "AL" THEN DE(I) = .100
8215  IF M$(I) = "AL" GOTO 8250
8220  IF M$(I) = "ST" THEN DE(I) = .283
8225  IF M$(I) = "ST" GOTO 8250
8230  IF M$(I) = "TI" THEN DE(I) = .160
8235  IF M$(I) = "TI" GOTO 8250
8240  IF M$(I) = "MA" THEN DE(I) = .064
8245  IF M$(I) = "MA" GOTO 8250
8248  PRINT "UNKNOWN MATERIAL-TRY AGAIN"
8249  GOTO 8112
8250  RETURN
9986  REM
9987  REM
```

```
9988  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 136,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

SAMPLE RUN

The following sample run illustrates how *CAD50* is used. Drawing CAD20.FRAME is recalled from disk, plotted on the screen, then node point coordinates and connectivities are listed. Following that, materials are assigned to the lines and the data listed again.

```
]RUN
CAD50
RECALLS A DRAWING
LISTS DATA
ADDS PROPERTIES TO LINES
REPLACES ON DISK
TYPE CONT TO CONTINUE

BREAK IN 40
]CONT
R=RECALL   C=CLEAR     L=LIST
F=FILE     E=END
P=PLOT     A=ADD PROPERTIES
COMMAND: R
RECALL
FILE NAME: CAD20.FRAME
DRIVE NUMBER: 1
R=RECALL   C=CLEAR     L=LIST
F=FILE     E=END
P=PLOT     A=ADD PROPERTIES
COMMAND: P
PLOT
```



```
R=RECALL    C=CLEAR     L=LIST
F=FILE      E=END
P=PLOT      A=ADD PROPERTIES
COMMAND: L
LIST
POINTS

1       1       0       0       0       7
2       2       500     0       0       7
3       3       500     200     0       7
4       4       0       200     0       7


LINES

1       1       2       7
        0       0

2       2       3       7
        0       0

3       3       4       7
        0       0

4       4       1       7
        0       0

5       1       3       7
        0       0

6       2       4       7
        0       0
```

```
R=RECALL    C=CLEAR      L=LIST
F=FILE      E=END
P=PLOT      A=ADD PROPERTIES
COMMAND: A
ADD PROPERTIES
AL=ALUMINUM    TI=TITANIUM     ST=STEEL
MA=MAGNESIUM   E=END
MATERIAL: AL
LINE: 1
AREA: 1.2
AL=ALUMINUM    TI=TITANIUM     ST=STEEL
MA=MAGNESIUM   E=END
MATERIAL: AL
LINE: 2
AREA: 1.2
AL=ALUMINUM    TI=TITANIUM     ST=STEEL
MA=MAGNESIUM   E=END
MATERIAL: AL
LINE: 3
AREA: 1.2
AL=ALUMINUM    TI=TITANIUM     ST=STEEL
MA=MAGNESIUM   E=END
MATERIAL: ST
LINE: 3
AREA: 1.2
AL=ALUMINUM    TI=TITANIUM     ST=STEEL
MA=MAGNESIUM   E=END
MATERIAL: MA
LINE: 5
AREA: 3.45
AL=ALUMINUM    TI=TITANIUM     ST=STEEL
MA=MAGNESIUM   E=END
MATERIAL: TI
LINE: 6
AREA: 5.432
AL=ALUMINUM    TI=TITANIUM     ST=STEEL
MA=MAGNESIUM   E=END
MATERIAL: E
R=RECALL    C=CLEAR      L=LIST
F=FILE      E=END
P=PLOT      A=ADD PROPERTIES
COMMAND: L
LIST
```

```
POINTS

1        1        0        0        0        7
2        2        500      0        0        7
3        3        500      200      0        7
4        4        0        200      0        7


LINES

1        1        2        7
AL       .1       1.2

2        2        3        7
AL       .1       1.2

3        3        4        7
ST       .283     1.2

4        4        1        7
         0        0

5        1        3        7
MA       .064     3.45

6        2        4        7
TI       .16      5.432
```

PROGRAM *CENTROID*

This is the first of our applications programs that operate on the drawings created and stored on disk by the previous *CAD* series of programs. *CENTROID* calculates and plots the location of the centroid (center of gravity) of a 3 dimensional structure. The normal procedure is to first create the 3 dimensional drawing using *CAD20*; update it if necessary with *CAD30;* and add material properties with *CAD50.* The drawing is, of course, stored on disk along the way. Program *CENTROID* then reads the final drawing, which completely defines the structure including material properties, and calculates the total weight and the x,y,z location of the centroid. It plots the location of the centroid on the screen and prints out its x,y,z coordinates on a printer.

PROGRAM LISTING

This program employs many of the familiar subroutines used in the *CAD* series plus some new ones. The input section, running from line 100-150, has been modified to include the command "G" in the menu. This calculates the location of the centroid and the total weight. Also added is the command "PG" which prints the centroid and weight information on a printer.

Subroutine 2000 calculates the total weight and the centroid location. The weight is obtained by summing the weights of the individual members. The centroid location is obtained by calculating the location of the centroid of the individual members and averaging.

Loop 2010-2180 carries out these tasks member by member. For each member, subroutine 4200 is called which locates the end node points. Lines 2050-2080 calculates the member's length using the square root of the sum of the squares. Line 2090 calculates the member's weight by multiplying its length by its cross sectional area and the material density, both of which should have been defined in *CAD50.* Line 2105 sums the individual member weights to arrive at the total system weight.

Lines 2120-2140 calculate the centroidal location of each bar. Since each has uniform density and cross sectional area along its length, the centroid of each individual bar must lie at its mid-point. The x,y,z location will therefore be the x,y,z location of the beginning node point of the bar plus half the distance to the end node.

Lines 2150-2170 sum the products of the member weights times the distance from the global axes to the member centroids. These quantities are then divided by the total weight in lines 2200-2220 to give the average distance from the coordinate axes, which is the definition of the centroid.

Lines 2260-2540 plot a star at the centroid's x,y location. Recall that the centroid was calculated in terms of real physical units, not Apple plotting units. Before the star can be plotted, it must be converted to Apple plotting units. This is accomplished by the call to subroutine 5400 in line 2280. This locates the center of the star, which is the centroid's location. The remainder of the star plotting statements define the corner coordinates of a star which is 8 plotting units across.

Subroutine 2600 prints the total system weight and the coordinates of the centroid on the printer.

```
10  PRINT "CENTROID"
11  PRINT "RECALLS A DRAWING"
13  PRINT "CALCULATES TOTAL WEIGHT"
14  PRINT "CENTROID LOCATION"
15  PRINT "PLOTS CENTROID LOCATION"
30  PRINT "TYPE CONT TO CONTINUE"
40  STOP
```

```
   50  HGR
   60  HCOLOR= 7
   70  GOSUB 9990
   80  DIM P(50),XP(50),YP(50),ZP(50),PC(50)
   90  DIM LB(25),LE(25),LC(25)
   95  DIM M$(25),AR(25),DE(25)
   96  REM
   97  REM
   98  REM
   99  REM
   100  REM  INPUT
➤  105  PRINT "R=RECALL"; TAB( 12);"PG=PRINT CENTROID"
➤  110  PRINT "G=CENTROID"; TAB( 12);"C=CLEAR"
   112  PRINT "P=PLOT"; TAB( 12);"E=END"
   115  INPUT "COMMAND: ";CK$
   118  IF CK$ = "E" GOTO 9998
   120  IF CK$ = "R" THEN  GOSUB 3000
➤  125  IF CK$ = "G" THEN  GOSUB 2000
   130  IF CK$ = "C" THEN  GOSUB 2900
   145  IF CK$ = "P" THEN  GOSUB 3500
➤  148  IF CK$ = "PG" THEN  GOSUB 2600
   150  GOTO 105
   1996  REM
   1997  REM
   1998  REM
   1999  REM
   2000  REM  CALCULATE CENTROID LOCATION
   2001  REM  AND TOTAL WEIGHT
   2005  PRINT "CENTROID LOCATION"
   2010  FOR I = 1 TO NL
   2020 LN = I
   2030  GOSUB 4200
   2040  REM  LINE LENGTH=Q
   2050 AA = XP(B) - XP(A)
   2060 BB = YP(B) - YP(A)
   2070 CC = ZP(B) - ZP(A)
   2080 Q =  SQR (AA * AA + BB * BB + CC * CC)
   2085  REM  LINE WEIGHT=W
   2090 W = Q * AR(I) * DE(I)
   2100  REM  TW=TOTAL WEIGHT
   2105 TW = TW + W
   2110  REM  LINE CENTROID=XC,YC,ZC
   2120 XC = XP(A) + AA / 2
   2130 YC = YP(A) + BB / 2
   2140 ZC = ZP(A) + CC / 2
   2150 SX = SX + W * XC
   2160 SY = SY + W * YC
   2170 SZ = SZ + W * ZC
   2180  NEXT I
   2190  REM  CENTROID LOCATION=CX,CY,CZ
   2200 CX = SX / TW
   2210 CY = SY / TW
   2220 CZ = SZ / TW
   2240  REM
```

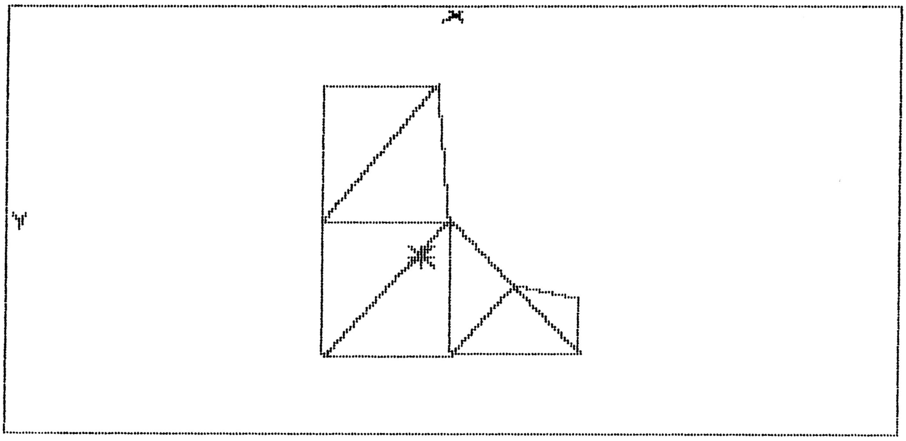
```
2250  REM  PLOT CENTROID LOCATION
2260 XR = CX
2270 YR = CY
2280  GOSUB 5400
2290 A = XT - 4
2300 AA = XT + 4
2310 B = YT
2320 BB = YT
2330  HPLOT A,B TO AA,BB
2400 A = XT
2410 AA = XT
2420 B = YT + 4
2430 BB = YT - 4
2440  HPLOT A,B TO AA,BB
2450 A = XT - 4
2460 AA = XT + 4
2470 B = YT - 4
2480 BB = YT + 4
2490  HPLOT A,B TO AA,BB
2500 A = XT + 4
2510 AA = XT - 4
2520 B = YT - 4
2530 BB = YT + 4
2540  HPLOT A,B TO AA,BB
2550  RETURN
2596  REM
2597  REM
2598  REM
2599  REM
2600  REM  PRINT CENTROID
2610  PRINT "PRINT CENTROID"
2620  PRINT D$;"PR#1"
2630  PRINT
2640  PRINT
2650  PRINT
2660  PRINT "TOTAL WEIGHT= ";TW
2670  PRINT
2680  PRINT "CX= ";CX
2690  PRINT "CY= ";CY
2700  PRINT "CZ= ";CZ
2710  PRINT D$;"PR#0"
2720  RETURN
2896  REM
2897  REM
2899  REM
2900  REM  CLEAR SCREEN
2905  PRINT "CLEAR"
2910  HGR
2915  GOSUB 9990
2929  RETURN
2996  REM
2997  REM
```

```
2998  REM
2999  REM
3000  REM  RECALL DRAWING FROM DISK
3005  PRINT "RECALL"
3010  INPUT "FILE NAME: ";F$
3020  INPUT "DRIVE NUMBER: ";DR
3030 D$ = "": REM  CTRL-D
3040  PRINT D$;"OPEN";F$;",D";DR
3050  PRINT D$;"READ";F$
3060  INPUT SF
3070  INPUT XO
3080  INPUT YO
3090  INPUT PX
3100  INPUT PY
3110  INPUT NP
3120  FOR I = 1 TO NP
3130  INPUT P(I)
3140  INPUT XP(I)
3150  INPUT YP(I)
3160  INPUT ZP(I)
3170  INPUT PC(I)
3180  NEXT I
3190  INPUT NL
3200  FOR I = 1 TO NL
3210  INPUT LB(I)
3220  INPUT LE(I)
3230  INPUT LC(I)
3235  INPUT M$(I)
3236  INPUT AR(I)
3237  INPUT DE(I)
3240  NEXT I
3250  PRINT D$;"CLOSE"
3260  RETURN
3496  REM
3497  REM
3498  REM
3499  REM
3500  REM  PLOT
3505  PRINT "PLOT"
3510  FOR I = 1 TO NP
3520 XR = XP(I)
3530 YR = YP(I)
3540  GOSUB 5400
3560  HCOLOR= PC(I)
3570  HPLOT XT,YT
3580  NEXT I
3590  FOR I = 1 TO NL
3595 LN = I
3600  GOSUB 4200
3610 XR = XP(A)
3620 YR = YP(A)
3630  GOSUB 5400
3640 AA = XT
3650 BB = YT
```

```
3660 XR = XP(B)
3670 YR = YP(B)
3680  GOSUB 5400
3690 CC = XT
3700 DD = YT
3710  HCOLOR= LC(I)
3720  HPLOT AA,BB TO CC,DD
3730  NEXT I
3740  RETURN
4140  RETURN
4196  REM
4197  REM
4198  REM
4199  REM
4200  REM  FIND POINTS
4205 MP = 0
4210  FOR K = 1 TO NP
4220  IF LB(LN) = P(K) THEN A = K
4230  IF LB(LN) = P(K) GOTO 4270
4240  NEXT K
4250  PRINT "CAN'T FIND POINT ";LB(LN)
4260 MP = 1
4265  GOTO 4330
4270  FOR K = 1 TO NP
4280  IF LE(LN) = P(K) THEN B = K
4290  IF LE(LN) = P(K) GOTO 4330
4300  NEXT K
4310  PRINT "CAN'T FIND POINT ";LE(LN)
4315 MP = 1
4330  RETURN
4796  REM
4797  REM
4798  REM
4799  REM
5400  REM  SCALE, ROTATE AND OFFSET PLOT COORDINATES
5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
9986  REM
9987  REM
9988  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 136,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

SAMPLE RUN

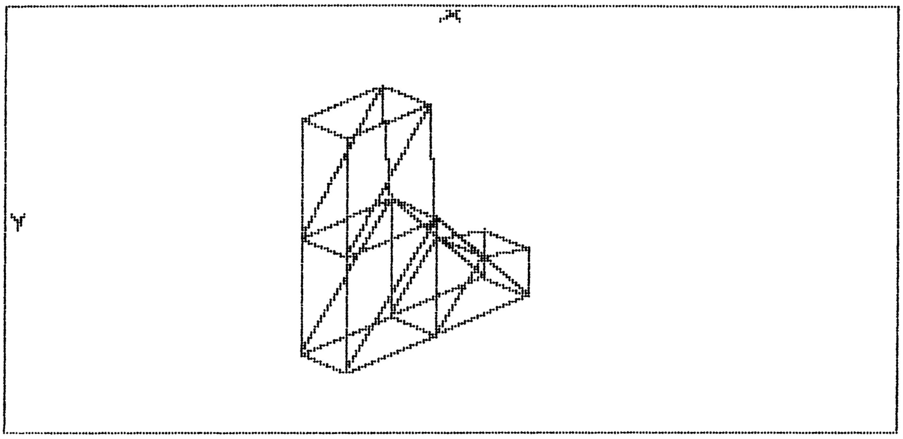
The computer plot below shows output from program *CENTROID*. The sequence of events was as follows: First drawing CAD20.TRUSS, which had been created earlier using *CAD20* and stored on disk, was recalled by *CAD50*. Member properties were added to the lines and the updated drawing was stored on disk under the name CAD50.TRUSS. Program *CENTROID* was then loaded, drawing CAD50.TRUSS was recalled from disk, and the evaluation and plotting of the centroid's coordinates was carried out.



```
TOTAL WEIGHT= 1284.31233

CX= 181.004528
CY= 152.153328
CZ= 0
```

PROGRAM *ISOVIEW*

This program provides a 3 dimensional view of a 3 dimensional drawing. An illustration is shown below.



To create this picture, the 2 dimensional drawing CAD20.TRUSS, which we created and stored earlier with *CAD20*, was updated by *CAD30* and changed to a 3 dimensional drawing. This was done by adding a second plane of members at $z = 100$ along with connecting members.

The new 3 dimensional version was stored on disk under the name CAD30.TRUSS3D. Listings of the drawing before and after the update are shown below:

LISTING OF DRAWING CAD20.TRUSS

This is a 2 dimensional truss created by program *CAD20:*

```
POINTS

1        1        500      0        0        7
2        4        500      180      0        7
3        2        250      0        0        7
4        5        250      200      0        7
5        3        0        0        0        7
6        9        0        200      0        7
7        6        125      300      0        7
8        7        100      400      0        7
9        8        0        400      0        7
```

```
LINES

1        1        4        7
2        1        2        7
3        2        4        7
4        4        5        7
5        2        5        7
6        2        3        7
7        3        9        7
8        5        9        7
9        3        5        7
10       5        6        7
11       9        6        7
12       8        9        7
13       6        8        7
14       6        7        7
15       7        8        7
```

LISTING OF CAD30.TRUSS3D

This is CAD20.TRUSS after updating by CAD30. A second plane has been added at z = 100 and connecting members were added:

```
POINTS

1        1        500      0        0        7
2        4        500      180      0        7
3        2        250      0        0        7
4        5        250      200      0        7
5        3        0        0        0        7
6        9        0        200      0        7
7        6        125      300      0        7
8        7        100      400      0        7
9        8        0        400      0        7
10       21       500      0        100      7
11       24       500      180      100      7
12       22       250      0        100      7
13       25       250      200      100      7
14       23       0        0        100      7
15       29       0        200      100      7
16       26       125      300      100      7
17       27       100      400      100      7
18       28       0        400      100      7
```

```
LINES

1         1         4         7
2         1         2         7
3         2         4         7
4         4         5         7
5         2         5         7
6         2         3         7
7         3         9         7
8         5         9         7
9         3         5         7
10        5         6         7
11        9         6         7
12        8         9         7
13        6         8         7
14        6         7         7
15        7         8         7
16        1         21        7
17        4         24        7
18        2         22        7
19        5         25        7
20        3         23        7
21        9         29        7
22        6         26        7
23        7         27        7
24        8         28        7
25        21        24        7
26        21        22        7
27        22        24        7
28        24        25        7
29        22        25        7
30        22        23        7
31        23        29        7
32        25        29        7
33        23        25        7
34        25        26        7
35        29        26        7
36        28        29        7
37        26        28        7
38        26        27        7
39        27        28        7
```

This drawing file was stored on disk. Then program *ISOVIEW* was loaded and CAD30.TRUSS3D was recalled from disk. The isometric view shown above was obtained by rotating the drawing through two angles: RX = 45 degrees and RY = 30 degrees.

PROGRAM LISTING

Program *ISOVIEW* employs the following main stages:

A - Input
B - Move
C - X axis rotation
D - Y axis rotation
E - Z axis rotation
F - Clear screen
G - Recall drawing
H - Plot
I - Find plots
J - Scale, rotate and offset
K - Calculate centroid
L - Z axis rotation matrix
M - X axis rotation matrix
N - Y axis rotation matrix
O - Coordinate transformation

In the input stage, the ''MOVE'' command in line 110 drops control to section B where the axis of rotation is asked for. Both the axis and the amount of rotation in degrees must be specified. Note that the axis of rotation is not the Apple's screen axis but the original physical axis used when the drawing data was originally input

As an illustration of how the rotation is carried out, assume RX as been typed, specifying a rotation about the real x axis. Control drops to subroutine 2100 where the amount of rotation is asked for in line 2110. This is converted to radians in line 2120.

Control then drops to subroutine 9600 where the components of the rotation transformation are calculated. The development of this transformation is a complicated affair. There are several books on the subject. The development used here is taken from *Graphic Software for Microcomputers* by B.J. Korites, also available from Kern Publications.

Loop 2140-2180 rotates all the points in the drawing, point-by-point. It does this be calling subroutine 9700 which performs the transformation using the previously defined matrix components. Note that a different set of matrix components will be involved for rotations about the other axes.

After the rotated positions of the points have been calculated, the screen is cleared by a call to subroutine 3500 in line 2200. There is no need to consider a separate rotation of lines. Because the lines are defined by the points, rotating the points automatically rotates the lines.

It is interesting to see how easy it was to develop this program. All we had to do was call upon the modular building blocks that were developed in earlier programs. We added a few new ones, but the amount of work required was greatly diminished by the modular approach we are using.

Before leaving this discussion, it should be mentioned that the drawing shown earlier had close to 50 members. The dimension statement in line 90 was temporarily increased to accommodate this.

```
10  PRINT "ISOVIEW"
11  PRINT "RECALLS A DRAWING"
12  PRINT "ROTATES IT IN"
13  PRINT "3 DIMENSIONS"
30  PRINT "TYPE CONT TO CONTINUE"
40  STOP
50  HGR
60  HCOLOR= 7
70  GOSUB 9990
80  DIM P(50),XP(50),YP(50),ZP(50),PC(50)
90  DIM LB(25),LE(25),LC(25)
96  REM
97  REM
98  REM
99  REM
```

A - Input

```
100  REM  INPUT
105  PRINT "R=RECALL"; TAB( 12);"P=PLOT"
110  PRINT "M=MOVE"; TAB( 12);"E=END"
115  INPUT "COMMAND: ";CK$
118  IF CK$ = "E" GOTO 9998
120  IF CK$ = "R" THEN  GOSUB 3000
125  IF CK$ = "M" THEN  GOSUB 2000
145  IF CK$ = "P" THEN  GOSUB 3500
150  GOTO 105
1996  REM
1997  REM
1998  REM
1999  REM
```

B - Move

```
2000  REM  MOVE
2010  PRINT "SELECT ROTATION"
2020  INPUT "RX,RY,RZ,E: ";M$
2025  IF M$ = "E" GOTO 2070
2030  IF M$ = "RX" THEN  GOSUB 2100
2040  IF M$ = "RY" THEN  GOSUB 2300
2050  IF M$ = "RZ" THEN  GOSUB 2500
2060  GOTO 2010
2066  REM
2067  REM
2068  REM
```

```
2069  REM
2070  RETURN
```

C - X axis rotation

```
2100  REM  X-AXIS ROTATION
2110  INPUT "X-ROTATION: ";RX
2120 RX = RX * 3.14159 / 180
2130  GOSUB 9600
2135  GOSUB 8000
2140  FOR I = 1 TO NP
2160  GOSUB 9700
2180  NEXT I
2190  GOSUB 2900
2200  GOSUB 3500
2206  REM
2207  REM
2208  REM
2209  REM
2210  RETURN
```

D - Y axis rotation

```
2300  REM  Y-ROTATION
2310  INPUT "Y-ROTATION: ";RY
2320 RY = RY * 3.14159 / 180
2330  GOSUB 9800
2340  FOR I = 1 TO NP
2360  GOSUB 9700
2380  NEXT I
2390  GOSUB 2900
2400  GOSUB 3500
2406  REM
2407  REM
2408  REM
2409  REM
2410  RETURN
```

E - Z axis rotation

```
2500  REM  Z-AXIS ROTATION
2510  INPUT "Z-AXIS ROTATION: ";RZ
2520 RZ = RZ * 3.14159 / 180
2530  GOSUB 9500
2540  FOR I = 1 TO NP
2560  GOSUB 9700
2580  NEXT I
2590  GOSUB 2900
2600  GOSUB 3500
2610  RETURN
2896  REM
2897  REM
2899  REM
```

F - Clear Screen

```
2900  REM  CLEAR SCREEN
2905  PRINT "CLEAR"
2910  HGR
2915  GOSUB 9990
2929  RETURN
2996  REM
2997  REM
```

G - Recall drawing from disk

```
2998  REM
2999  REM
3000  REM  RECALL DRAWING FROM DISK
3005  PRINT "RECALL"
3010  INPUT "FILE NAME: ";F$
3020  INPUT "DRIVE NUMBER: ";DR
3030 D$ = "": REM  CTRL-D
3040  PRINT D$;"OPEN";F$;",D";DR
3050  PRINT D$;"READ";F$
3060  INPUT SF
3070  INPUT XO
3080  INPUT YO
3090  INPUT PX
3100  INPUT PY
3110  INPUT NP
3120  FOR I = 1 TO NP
3130  INPUT P(I)
3140  INPUT XP(I)
3150  INPUT YP(I)
3160  INPUT ZP(I)
3170  INPUT PC(I)
3180  NEXT I
3190  INPUT NL
3200  FOR I = 1 TO NL
3210  INPUT LB(I)
3220  INPUT LE(I)
3230  INPUT LC(I)
3240  NEXT I
3250  PRINT D$;"CLOSE"
3260  RETURN
3496  REM
3497  REM
3498  REM
3499  REM
```

Note that line properties are not included here although they could be.

H - Plot drawing

```
3500  REM  PLOT
3505  PRINT "PLOT"
3510  FOR I = 1 TO NP
3520 XR = XP(I)
3530 YR = YP(I)
3540  GOSUB 5400
3560  HCOLOR= PC(I)
3570  HPLOT XT,YT
3580  NEXT I
3590  FOR I = 1 TO NL
3595 LN = I
3600  GOSUB 4200
3610 XR = XP(A)
3620 YR = YP(A)
3630  GOSUB 5400
3640 AA = XT
3650 BB = YT
3660 XR = XP(B)
3670 YR = YP(B)
3680  GOSUB 5400
```

```
3690 CC = XT
3700 DD = YT
3710  HCOLOR= LC(I)
3720  HPLOT AA,BB TO CC,DD
3730  NEXT I
3740  RETURN
4196  REM
4197  REM
4198  REM
4199  REM
```

I-Find points

```
4200  REM  FIND POINTS
4205 MP = 0
4210  FOR K = 1 TO NP
4220  IF LB(LN) = P(K) THEN A = K
4230  IF LB(LN) = P(K) GOTO 4270
4240  NEXT K
4250  PRINT "CAN'T FIND POINT ";LB(LN)
4260 MP = 1
4265  GOTO 4330
4270  FOR K = 1 TO NP
4280  IF LE(LN) = P(K) THEN B = K
4290  IF LE(LN) = P(K) GOTO 4330
4300  NEXT K
4310  PRINT "CAN'T FIND POINT ";LE(LN)
4315 MP = 1
4330  RETURN
4796  REM
4797  REM
4798  REM
4799  REM
```

J - Scale, rotate and offset

```
5400  REM  SCALE, ROTATE AND OFFSET PLOT COORDINATES
5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
7996  REM
7997  REM
7998  REM
7999  REM
```

K - Calculate centroid location

See program CENTROID for an explanation of how the geometric center is calculated. We use it here as a convenient center of rotation.

```
8000  REM  CALCULATE GEOMETRIC CENTER
8001  REM  WHICH WILL BE CENTER OF ROTATION
8010  FOR I = 1 TO NP
8005 SX = 0:SY = 0:SZ = 0
8010  FOR I = 1 TO NP
8020 SX = SX + XP(I)
8030 SY = SY + YP(I)
8040 SZ = SZ + ZP(I)
8050  NEXT I
8060 XC = SX / NP
8070 YC = SY / NP
8080 ZC = SZ / NP
8086  REM
8087  REM
```

```
8088  REM
8089  REM
8090  RETURN
```

L - Z axis rotation matrix

```
9500  REM  Z-AXIS ROTATION MATRIX
9510 C(1,1) =  COS (RZ)
9520 C(1,2) =  SIN (RZ)
9530 C(1,3) = 0
9540 C(2,1) =  -  SIN (RZ)
9550 C(2,2) =  COS (RZ)
9560 C(2,3) = 0
9570 C(3,1) = 0
9580 C(3,2) = 0
9590 C(3,3) = 1
9595  RETURN
```

M - X axis rotation matrix

```
9600  REM  X-AXIS ROTATION MATRIX
9610 C(1,1) = 1
9620 C(1,2) = 0
9630 C(1,3) = 0
9640 C(2,1) = 0
9650 C(2,2) =  COS (RX)
9660 C(2,3) =  SIN (RX)
9670 C(3,1) = 0
9680 C(3,2) =  -  SIN (RX)
9690 C(3,3) =  COS (RX)
9695  RETURN
9696  REM
9697  REM
9698  REM
9699  REM
```

O - Coordinate transformation

```
9700  REM  COORDINATE TRANSFORMATION
9705 A1 = XP(I) - XC
9706 A2 = YP(I) - YC
9707 A3 = ZP(I) - ZC
9710 XP(I) = XC + A1 * C(1,1) + A2 * C(2,1) + A3 * C(3,1)
9720 YP(I) = YC + A1 * C(1,2) + A2 * C(2,2) + A3 * C(3,2)
9730 ZP(I) = ZC + A1 * C(1,3) + A2 * C(2,3) + A3 * C(3,3)
9740  RETURN
9778  REM
9796  REM
9797  REM
9799  REM
```

N - Y axis rotation matrix

```
9800  REM  Y-AXIS ROTATION MATRIX
9810 C(1,1) =  COS (RY)
9820 C(1,2) = 0
9830 C(1,3) =  -  SIN (RY)
9840 C(2,1) = 0
9850 C(2,2) = 1
9860 C(2,3) = 0
9870 C(3,1) =  SIN (RY)
9880 C(3,2) = 0
9890 C(3,3) =  COS (RY)
9895  RETURN
9986  REM
9987  REM
```

```
9988  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 136,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

SUMMARY

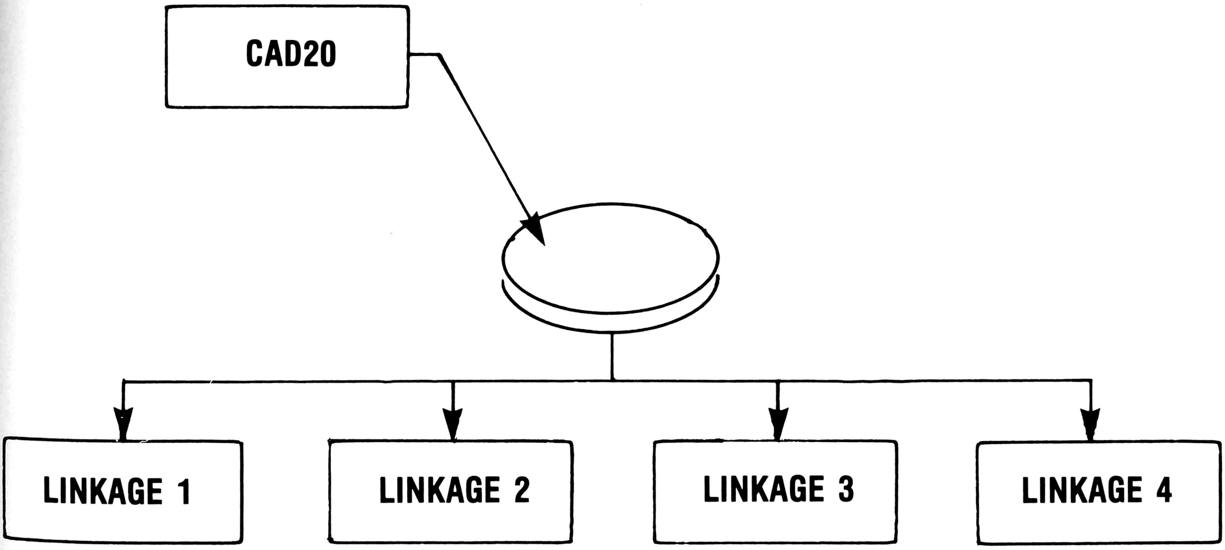
Before concluding this chapter on CAD/CAM software, a few things should be mentioned: First, it is recognized that the programs developed here may not do exactly what you want. For example, not all programs will list data on a printer; *CAD50* does not add physical properties to points, *ISOVIEW* does not read and plot drawing files that include physical properties. The list of combinations of things that could be included in a program is endless. Unfortunately, if a single program did everything we wanted it to, it wouldn't fit in a micro's memory.

Our intention is to give you the programming techniques and the basic building blocks that you need to modify and combine these programs so as to construct programs of your own that will be tailor made to suit your own particular applications. By using the modular approach we have used in these programs, together with the menu driven program structure, you should be able to custom write software of your own with little difficulty.

Chapter 3: Mechanisms

Introduction, *3-1*

LINKAGE1: Static simulation of the motion of a four bar linkage, *3-3*
LINKAGE2: Static simulation of the motion of a four bar linkage with a follower bar, *3-9*
LINKAGE3: Dynamic simulation of the motion of a four bar linkage, *3-16*
LINKAGE4: Dynamic simulation of the motion of a slider crank, *3-19*



CAD20
LINKAGE 1
LINKAGE 2
LINKAGE 3
LINKAGE 4

3-1 INTRODUCTION

Mechanisms

In this section we will develop a program to simulate the motion of a four bar linkage. The four bar linkage is fundamental to the study of mechanisms since more complex mechanisms can be represented as an assemblage of four bars. The programs and techniques we will be developing here can be easily extrapolated to simulate the motion of practically any mechanism.

As we will see, computer simulation of mechanism motion, especially with a graphical animated display of motion, is a tremendously valuable tool for the study of mechanisms since it allows us to visually observe what is predicted by equations. For example, some four bar mechanisms will bind up under certain conditions. While this phenomenon can be predicted analytically, and can be visualized on paper after enough sketching and thinking, it can be easily observed on the screen of a microcomputer with the proper software.

A side effect of computer simulation of mechanism motion is that when we write the simulation software, we are *forced* to understand the workings of the mechanism in detail. So, not only does computer simulation help us understand how a mechanism behaves phenomenologically, the programming process helps us understand the mechanism from an analytical viewpoint as well.

In all the programs which we shall develop in this chapter, the input data describing the mechanism will be assumed to have been created and stored on disk by one of the *CAD* programs developed in the last chapter. These simulation programs will therefore be termed *applications* programs since they carry out specific applications on the data stored in the drawing data base.

There aren't many points and lines in a four bar linkage or a slider crank and it isn't really necessary to store the data on disk; we could just as easily input the data at the start of each of these linkage programs. However, we want to emphasize the CAD/CAM approach which could very well be a necessity in the study of more complex systems.

We will develop four programs in this chapter: *LINKAGE1* displays motion of a four bar linkage with the crank angle manually keyed in by the user; *LINKAGE2* displays motion of a four bar linkage with a follower bar. The image of the follower bar remains on the screen as the linkage moves. This aids in designing mechanisms of this type since the motion of the follower bar as a function of crank angle is often an important design function; *LINKAGE3* displays motion of a four bar linkage dynamically; *LINKAGE4* lists the follower position vs crank angle on a printer for the four bar with follower.

The geometry we will by studying is shown in Figure 3-1. This figure shows the basic four bar linkage. G1 is considered to be the crank angle. G2 is the follower angle. The relevant equations are displayed beneath the sketch.

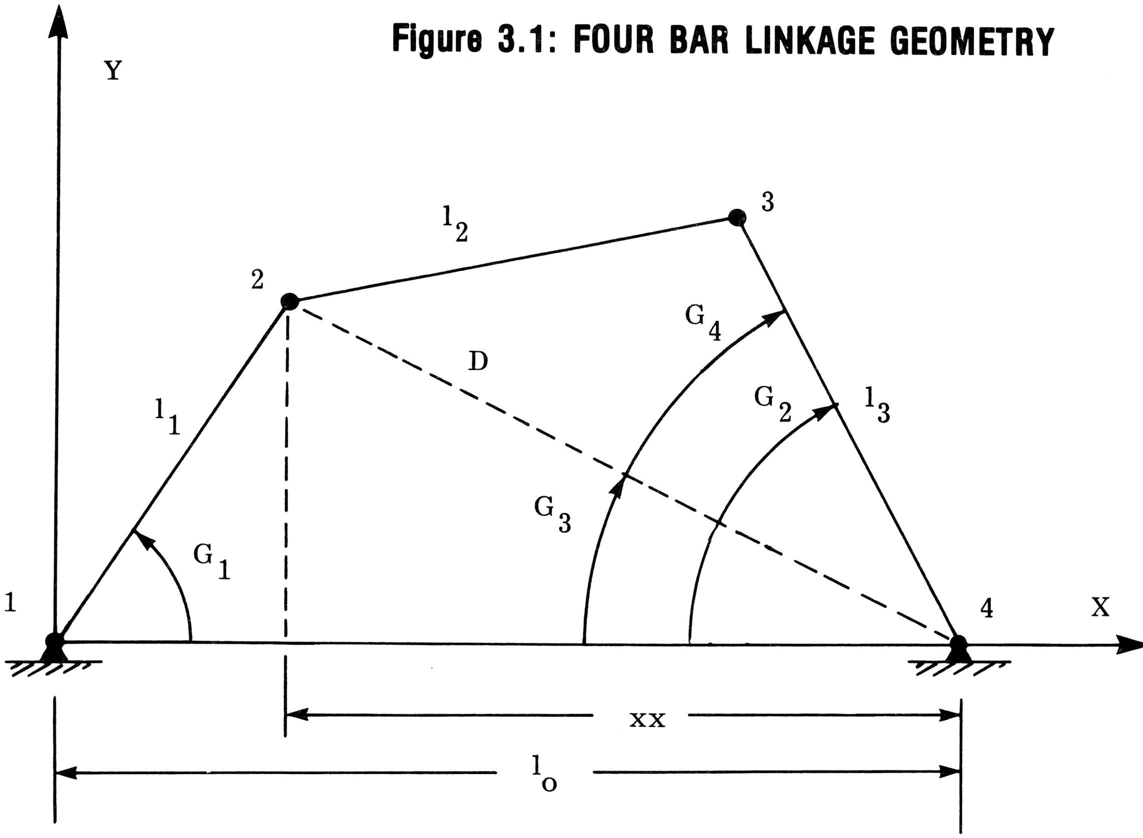


Figure 3.1: FOUR BAR LINKAGE GEOMETRY

xp(1), yp(1).....xp(4), yp(4) are starting coordinates

x1, y1..........x4, y4 are coordinates after motion starts

A = xp(2) - xp(1)

B = yp(2) - yp(1)

l_1 = SQR(A*A + B*B)

similarly for the other bar lengths

x2 = l_1*sin(G1)

y2 = l_1*cos(G1)

$$xx = l_0 - (x2 - x1)$$
$$yy = y2 - y1$$

$$G3 = ATN(yy/xx)$$
$$D = SQR(xx*xx + yy*yy)$$

$$G4 = (D*D + l_3*l_3 - l_2*l_2)/(2*D*l_3) - \quad \text{(Law of Cosines)}$$
$$= ATN(A/SQR(-A*A+1)) + 1.5708 \quad (\text{from } \sin^2 + \cos^2 = 1)$$

$$G2 = G3 + G4$$

$$x3 = l_0 - l_3*\cos(G2)$$
$$y3 = l_3*\sin(G2)$$

PROGRAM *LINKAGE1*

The coordinates of four node points and bars are read from disk in lines 120-410. As mentioned earlier, the geometry of the mechanism is assumed to have been created earlier with one of the *CAD* programs and stored on disk.

The sequence of the nodes on the disk file is important. They must be in the same sequence as shown in Figure 3-1.

Lines 430-550 calculate the bar lengths from the initial node coordinates. Lines 560-600 calculate the initial crank angle, G1. A new crank angle can be input in line 610. Lines 612 and 613 erase the original picture in preparation for the calculation of a new position defined by a new crank angle, G1.

The erasing is done by subroutine 730. The position of the various nodes - (x1,y1), (x2,y2).....(x4,y4) are transformed to plotting values by subroutine 5400. The scale factor, offset values and coordinate angles were passed along on the drawing file and are read in lines 170-210. Each node has its coordinates transformed by subroutine 5400 and the transformed values are plotted. The original physical values, as stored on the disk drawing file, remain unchanged.

It is important to recognize that we are reading real physical values from the disk, but we are plotting transformed values. The transforming is done in subroutine 5400.

Line 614 resets the color to white. G1 is converted from degrees to radians in line 620. Lines 625-670 calculate G2, the follower angle, according to the equations written earlier. Lines 710-720 calculate the position of node 3. The new position is then plotted by a call to 730.

```
10  PRINT "LINKAGE1"
20  PRINT "PLOTS AND PRINTS"
30  PRINT "PATHS OF NODES IN"
40  PRINT "A 4 BAR LINKAGE"
50  PRINT "READS LINKAGE GEOMETRY"
60  PRINT "FROM DISK"
70  PRINT "TYPE CONT TO CONTINUE"
80  STOP
90  HGR
100  HCOLOR= 7
110  GOSUB 9990
115  DIM P(10),XP(10),YP(10),ZP(10),PC(10)
116  DIM LB(10),LE(10),LC(10)
120  INPUT "FILE NAME: ";F$
130  INPUT "DRIVE NUMBER: ";DR
140 D$ = "": REM  CTRL-D
150  PRINT D$;"OPEN";F$;",D";DR
160  PRINT D$;"READ";F$
170  INPUT SF
180  INPUT XO
190  INPUT YO
200  INPUT PX
210  INPUT PY
220  INPUT NP
230  FOR I = 1 TO NP
240  INPUT P(I)
250  INPUT XP(I)
260  INPUT YP(I)
270  INPUT ZP(I)
280  INPUT PC(I)
290  NEXT I
300  INPUT NL
310  FOR I = 1 TO NL
320  INPUT LB(I)
330  INPUT LE(I)
340  INPUT LC(I)
350  NEXT I
405  PRINT D$;"CLOSE"
412 X1 = XP(1)
413 Y1 = YP(1)
414 X2 = XP(2)
415 Y2 = YP(2)
416 X3 = XP(3)
417 Y3 = YP(3)
418 X4 = XP(4)
```

```
  419 Y4 = YP(4)
  420  GOSUB 730
➤ 430  REM  CALCULATE BAR LENGTHS
  440 A = X2 - X1
  450 B = Y2 - Y1
  460 L1 =  SQR (A * A + B * B)
  470 A = X3 - X2
  480 B = Y3 - Y2
  490 L2 =  SQR (A * A + B * B)
  500 A = X4 - X3
  510 B = Y4 - Y3
  520 L3 =  SQR (A * A + B * B)
  530 A = X4 - X1
  540 B = Y4 - Y1
  550 L0 =  SQR (A * A + B * B)
➤ 560  REM  CALCULATE AND PRINT G1
  565 XX = L0 - (X2 - X1)
  570 A = Y2 - Y1
  575 B = L0 - XX
  576  IF B = 0 GOTO 578
  577  GOTO 582
  578  IF A > 0 THEN G1 = 3.14159 / 2
  579  IF A < 0 THEN G1 =  - 3.14159 / 2
  580  GOTO 590
  582 G1 =  ATN (A / B)
  590 B = G1 * 180 / 3.14159
  600  PRINT "G1(DEGREES)= ";B
  610  INPUT "SELECT G1(DEGREES): ";G1
➤ 611  REM  ERASE LAST PICTURE
  612  HCOLOR= 0
  613  GOSUB 730
  614  HCOLOR= 7
  620 G1 = G1 * 3.14159 / 180
➤ 625  REM  CALCULATE G2
  626 Y2 = L1 *  SIN (G1)
  627 X2 = L1 *  COS (G1)
  628 XX = L0 - (X2 - X1)
  629 YY = Y2
  630 G3 =  ATN (YY / XX)
  640 D =  SQR (XX * XX + YY * YY)
  641  IF D > (L2 + L3) THEN  GOSUB 1000
  650 A = (D * D + L3 * L3 - L2 * L2) / (2 * D * L3)
  651  IF A = 1 THEN  GOSUB 1050
  652  IF A > 1 THEN  GOSUB 2000
  660 G4 =  -  ATN (A /  SQR ( - A * A + 1)) + 1.5708
  670 G2 = G3 + G4
➤ 680  REM  CALCULATE NEW COORDS
  710 X3 = L0 - L3 *  COS (G2)
  720 Y3 = L3 *  SIN (G2)
  725  GOSUB 730
  726  GOTO 590
  730  REM  TRANSFORM AND PLOT
  740 XR = X1
  750 YR = Y1
```

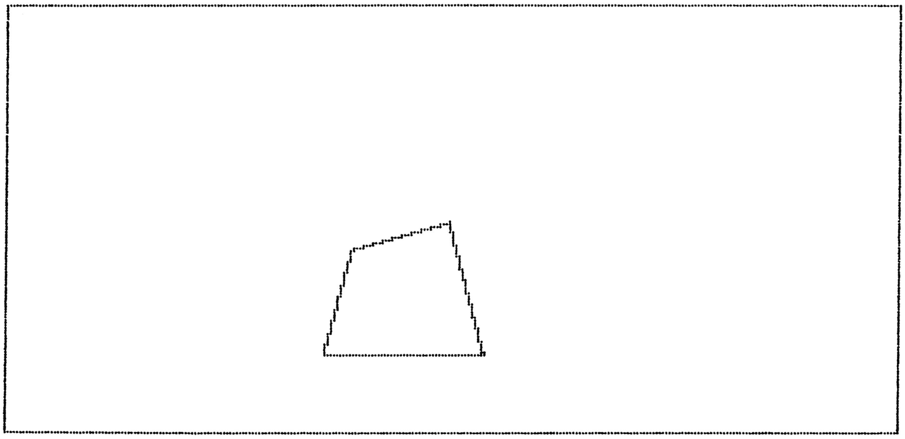
```
760  GOSUB 5400
770  HPLOT XT,YT
780 XR = X2
790 YR = Y2
800  GOSUB 5400
810  HPLOT  TO XT,YT
820 XR = X3
830 YR = Y3
840  GOSUB 5400
850  HPLOT  TO XT,YT
~~855  GOTO 940~~
860 XR = X4
870 YR = Y4
880  GOSUB 5400
890  HPLOT  TO XT,YT
900 XR = X1
910 YR = Y1
920  GOSUB 5400
930  HPLOT  TO XT,YT
940  RETURN
1000  PRINT "D>(L2+L3)-LINKAGE BOUND"
1010  PRINT "TRY ANOTHER ANGLE"
1020  GOTO 590
1025  RETURN
1050  PRINT "A=1 - LINKAGE BOUND"
1060  PRINT "TRY ANOTHER ANGLE"
1070  GOTO 590
1080  RETURN
2000  PRINT "A>1 - LINKAGE BOUND"
2010  PRINT "TRY ANOTHER ANGLE"
2020  GOTO 590
2030  RETURN
5400  REM  SCALE,ROTATE AND OFFSET PLOT COORDS
5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

Lines 641-652 deserve some comment since they illustrate nicely one of the primary advantages of observing the response of a system like this on a computer. Looking at line 641, it is clear that if D is greater than the sum of l1 plus l2, then the linkage will bind up. This situation could occur with certain bar lengths and crank angles. If it does happen, it will be detected by line 641 and an error message will be printed by subroutine 1000. Control will return to line 590 where another angle may be input.

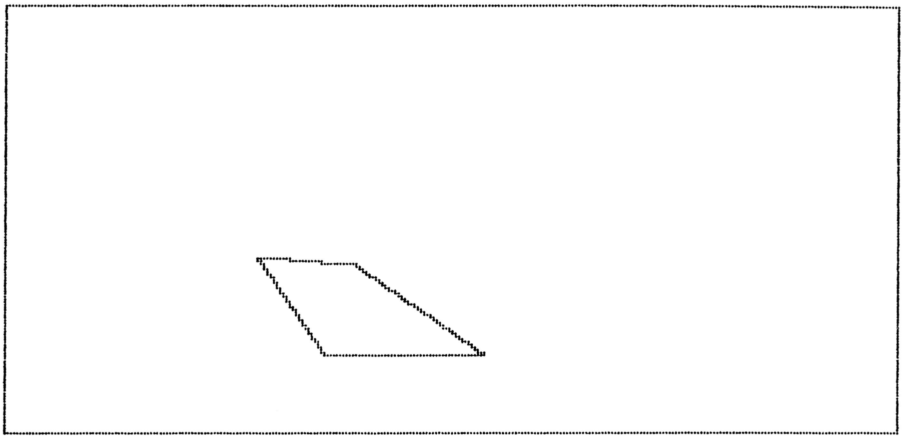
The situation detected by lines 650-652 is not so obvious. What happens here is that if D is less than l3 minus l2, the mechanism will again bind up. Actually, the statement that it will "bind up" is not quite correct. The conditions D greater than l1 plus l2 and D less than l3 minus l2 are impossible. The mechanism binds up when D equals these values.

This situation is neatly illustrated by running the program with gradual increases in the crank angle. The binding conditions are readily observed. The following sample run illustrates:

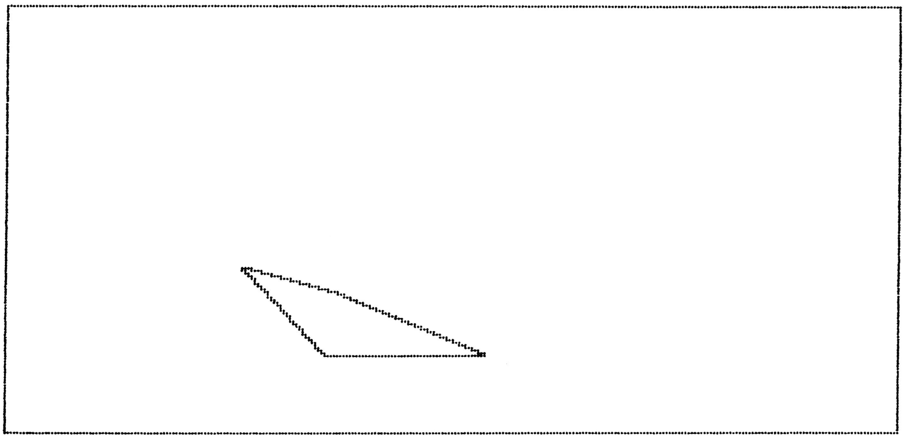
```
]RUN
LINKAGE1
PLOTS AND PRINTS
PATHS OF NODES IN
A 4 BAR LINKAGE
READS LINKAGE GEOMETRY
FROM DISK
TYPE CONT TO CONTINUE

BREAK IN 80
]CONT
FILE NAME: FOURBAR.PIC
DRIVE NUMBER: 1
G1(DEGREES)= 75.9638207
```



```
SELECT G1(DEGREES):  120
G1(DEGREES)=  120
```



```
SELECT G1(DEGREES):  129
G1(DEGREES)=  129
```



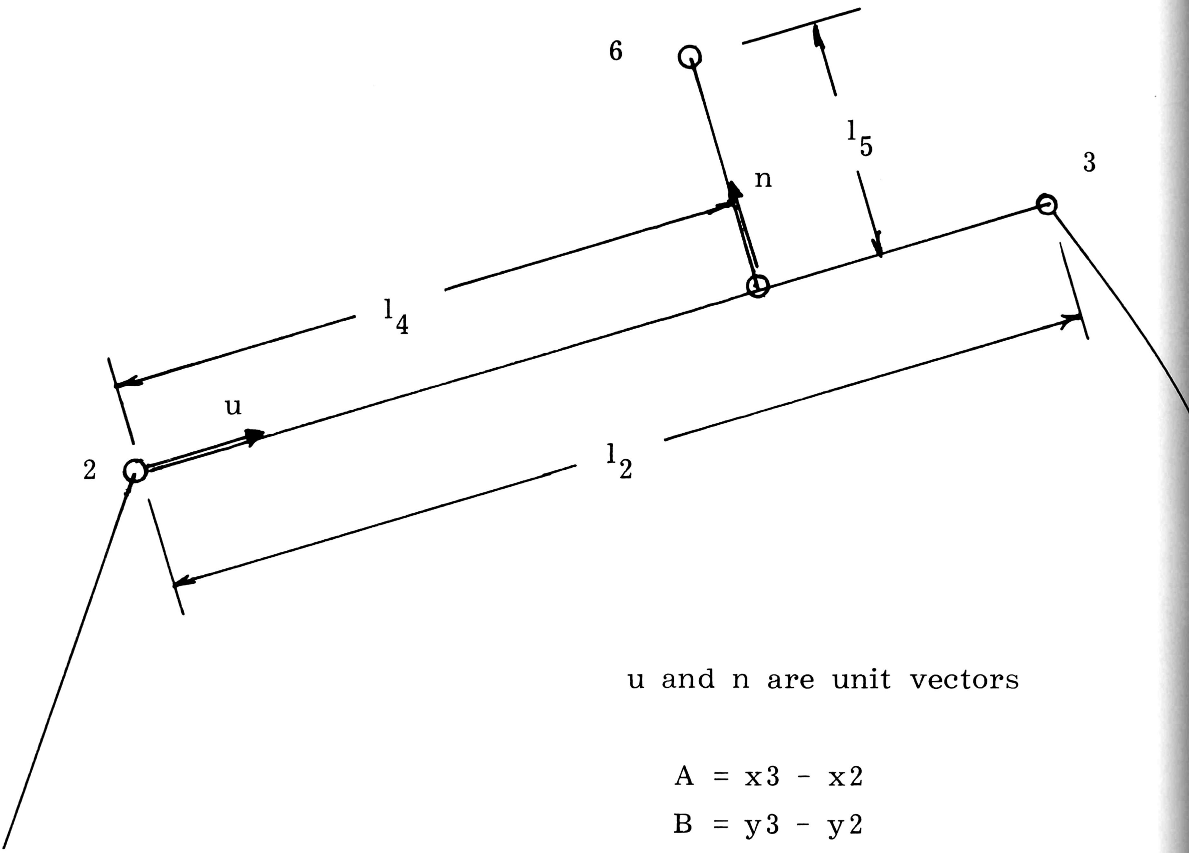
```
SELECT G1(DEGREES):  129.5
G1(DEGREES)=  129.5
SELECT G1(DEGREES):  130
D>(L2+L3)-LINKAGE BOUND
TRY ANOTHER ANGLE
```

LINKAGE2

PROGRAM *LINKAGE2*

This program is similar to *LINKAGE1* except that a fifth bar has been added. This is a follower bar and the trajectory of its tip is of interest in the design of mechanisms.

The geometry we will use to calculate the position of the follower is shown in Figure 3-2. The relevant equations follow. We will use vectors to calculate the position of the various node points.



u and n are unit vectors

$A = x3 - x2$

$B = y3 - y2$

$ux = A/l_2$

$uy = B/l_2$

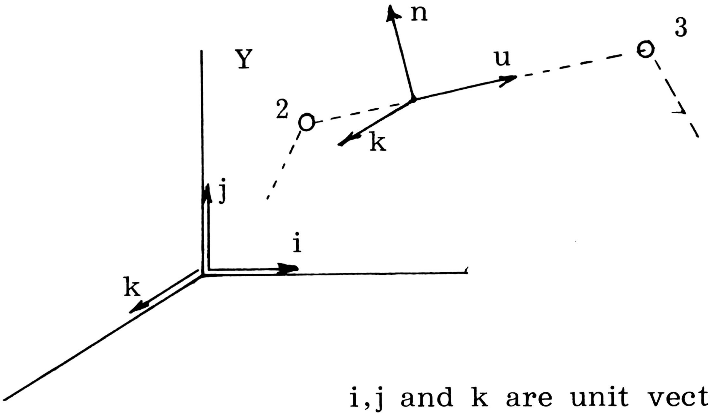
Figure 3.2: FOUR BAR MECHANISM WITH FOLLOWER

$$x5 = x2 + ux*l_4$$

$$y5 = y2 + uy*l_4$$

LINKAGE2

The unit vector **n** is determined by taking the vector product (cross product) of **k**, which is a unit vector in the direction of the z axis, with **u**, a unit vector along the bar from node 2 to 3. This assumes that the follower bar is perpendicular to the bar running from node 2 to 3 since the vector product of two unit vectors produces a third vector which is perpendicular to the plane defined by the first two. That is, the bar 2-3 lies in the x-y plane, **k** is perpendicular to that plane, so the vector product produces a third vector which is perpendicular to both bar 2-3 and the z axis. Since **u** and **k** are unit vectors, and since the angle between them is 90 degrees, the result of the vector product, **n**, is also a unit ventor.



i,j and k are unit vectors in the x,y,z directions

$$\begin{aligned} n &= kxu \qquad \text{(vector product)} \\ &= kx(uxi + uyj) \\ &= ux(kxi) = uy(kxj) \\ &= uxj + uy(-i) \end{aligned}$$

$$n = nxi + nyj$$

therefore:

$$nx = ux$$

$$ny = -uy$$

PROGRAM LISTING

The program has been written so that the image of the follower bar remains on the screen after each position has been plotted. This has been done so that the trajectory of the follower can be studied. The program structure is similar to *LINKAGE1* except for the addition of the follower bar and related equations.

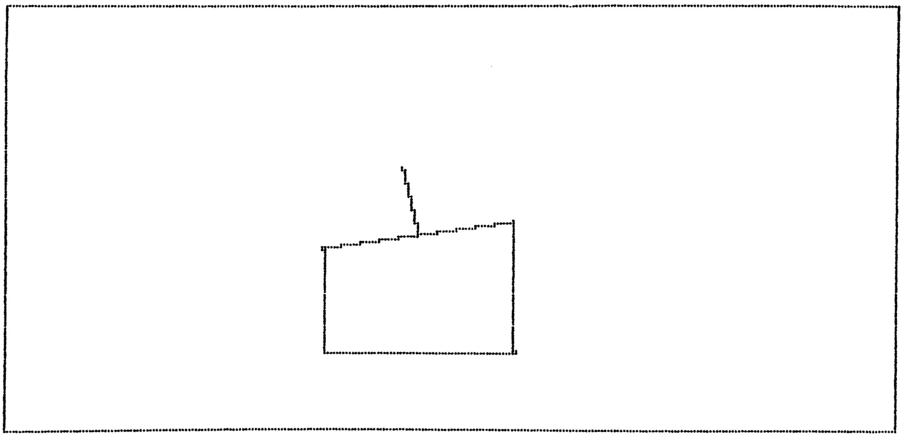
```
10  PRINT "LINKAGE2"
20  PRINT "PLOTS AND PRINTS"
30  PRINT "PATHS OF NODES IN"
40  PRINT "A 4 BAR LINKAGE"
42  PRINT "WITH FOLLOWER BAR"
50  PRINT "READS LINKAGE GEOMETRY"
60  PRINT "FROM DISK"
70  PRINT "TYPE CONT TO CONTINUE"
80  STOP
90  HGR
100  HCOLOR= 7
110  GOSUB 9990
115  DIM P(10),XP(10),YP(10),ZP(10),PC(10)
116  DIM LB(10),LE(10),LC(10)
117  DIM T1(10),T2(10),T3(10),TC(10)
120  INPUT "FILE NAME: ";F$
130  INPUT "DRIVE NUMBER: ";DR
140 D$ = "": REM  CTRL-D
150  PRINT D$;"OPEN";F$;",D";DR
160  PRINT D$;"READ";F$
170  INPUT SF
180  INPUT XO
190  INPUT YO
200  INPUT PX
210  INPUT PY
220  INPUT NP
230  FOR I = 1 TO NP
240  INPUT P(I)
250  INPUT XP(I)
260  INPUT YP(I)
270  INPUT ZP(I)
280  INPUT PC(I)
290  NEXT I
300  INPUT NL
310  FOR I = 1 TO NL
320  INPUT LB(I)
330  INPUT LE(I)
340  INPUT LC(I)
350  NEXT I
405  PRINT D$;"CLOSE"
412 X1 = XP(1)
413 Y1 = YP(1)
414 X2 = XP(2)
415 Y2 = YP(2)
```

```
416 X3 = XP(3)
417 Y3 = YP(3)
418 X4 = XP(4)
419 Y4 = YP(4)
420 X5 = XP(5)
421 Y5 = YP(5)
422 X6 = XP(6)
423 Y6 = YP(6)
425  GOSUB 730
426 XR = X5
427 YR = Y5
428  GOSUB 5400
429  HPLOT XT,YT
430 XR = X6
431 YR = Y6
432  GOSUB 5400
433  HPLOT  TO XT,YT
435  REM  CALCULATE BAR LENGTHS
440 A = X2 - X1
450 B = Y2 - Y1
460 L1 =  SQR (A * A + B * B)
470 A = X3 - X2
480 B = Y3 - Y2
490 L2 =  SQR (A * A + B * B)
500 A = X4 - X3
510 B = Y4 - Y3
520 L3 =  SQR (A * A + B * B)
530 A = X4 - X1
540 B = Y4 - Y1
550 L0 =  SQR (A * A + B * B)
551 A = X5 - X2
552 B = Y5 - Y2
553 L4 =  SQR (A * A + B * B)
554 A = X6 - X5
555 B = Y6 - Y5
556 L5 =  SQR (A * A + B * B)
560  REM  CALCULATE AND PRINT G1
565 XX = L0 - (X2 - X1)
570 A = Y2 - Y1
575 B = L0 - XX
576  IF B = 0 GOTO 578
577  GOTO 582
578  IF A > 0 THEN G1 = 3.14159 / 2
579  IF A < 0 THEN G1 =  - 3.14159 / 2
580  GOTO 590
582 G1 =  ATN (A / B)
590 B = G1 * 180 / 3.14159
600  PRINT "G1(DEGREES)= ";B
610  INPUT "SELECT G1(DEGREES): ";G1
611  REM  ERASE LAST PICTURE
612  HCOLOR= 0
613  GOSUB 730
614  HCOLOR= 7
620 G1 = G1 * 3.14159 / 180
```

```
625  REM  CALCULATE G2
626 Y2 = L1 *  SIN (G1)
627 X2 = L1 *  COS (G1)
628 XX = L0 - (X2 - X1)
629 YY = Y2
630 G3 =  ATN (YY / XX)
640 D =  SQR (XX * XX + YY * YY)
641  IF D > (L2 + L3) THEN  GOSUB 1000
650 A = (D * D + L3 * L3 - L2 * L2) / (2 * D * L3)
651  IF A = 1 THEN  GOSUB 1050
652  IF A > 1 THEN  GOSUB 2000
660 G4 =  -  ATN (A /  SQR ( - A * A + 1)) + 1.5708
670 G2 = G3 + G4
680  REM  CALCULATE NEW COORDS
710 X3 = L0 - L3 *  COS (G2)
720 Y3 = L3 *  SIN (G2)
725  GOSUB 3000
726  GOSUB 730
727  GOTO 590
730  REM  TRANSFORM AND PLOT
740 XR = X1
750 YR = Y1
760  GOSUB 5400
770  HPLOT XT,YT
780 XR = X2
790 YR = Y2
800  GOSUB 5400
810  HPLOT  TO XT,YT
820 XR = X3
830 YR = Y3
840  GOSUB 5400
850  HPLOT  TO XT,YT
860 XR = X4
870 YR = Y4
880  GOSUB 5400
890  HPLOT  TO XT,YT
900 XR = X1
910 YR = Y1
920  GOSUB 5400
930  HPLOT  TO XT,YT
980  RETURN
1000  PRINT "D>(L2+L3)-LINKAGE BOUND"
1010  PRINT "TRY ANOTHER ANGLE"
1020  GOTO 590
1025  RETURN
1050  PRINT "A=1 - LINKAGE BOUND"
1060  PRINT "TRY ANOTHER ANGLE"
1070  GOTO 590
1080  RETURN
2000  PRINT "A>1 - LINKAGE BOUND"
2010  PRINT "TRY ANOTHER ANGLE"
2020  GOTO 590
2030  RETURN
3000  REM  PLOT FOLLOWER PATH
```

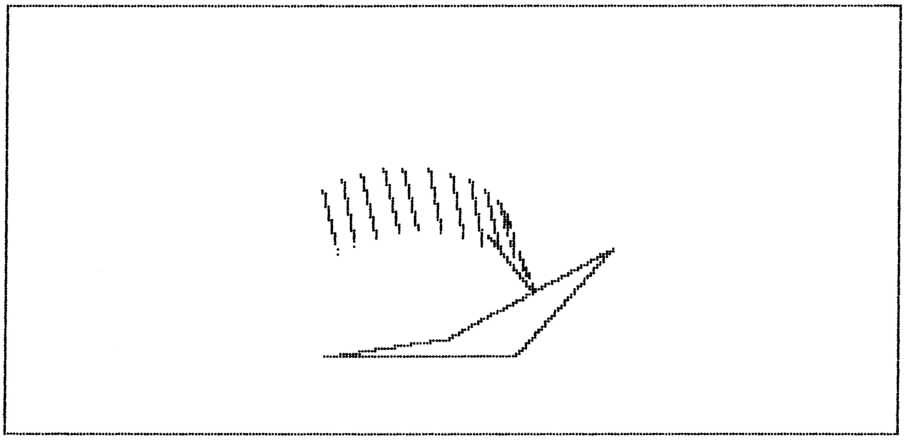
```
3010 A = X3 - X2
3020 B = Y3 - Y2
3030 UX = A / L2
3040 UY = B / L2
3050 X5 = X2 + UX * L4
3060 Y5 = Y2 + UY * L4
3061 XR = X5
3062 YR = Y5
3063  GOSUB 5400
3064  HPLOT XT,YT
3070 NX =  - UY
3080 NY = UX
3130 X6 = X5 + L5 * NX
3140 Y6 = Y5 + L5 * NY
3150 XR = X6
3160 YR = Y6
3170  GOSUB 5400
3180  HPLOT  TO XT,YT
3190  RETURN
5400  REM  SCALE,ROTATE AND OFFSET PLOT COORDS
5410 XT = X0 + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = Y0 + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

```
]RUN
LINKAGE2
PLOTS AND PRINTS
PATHS OF NODES IN
A 4 BAR LINKAGE
WITH FOLLOWER BAR
READS LINKAGE GEOMETRY
FROM DISK
TYPE CONT TO CONTINUE
```



LINKAGE2

```
BREAK IN 80
]CONT
FILE NAME: FOLLOWER
DRIVE NUMBER: 1
G1(DEGREES)= 90
SELECT G1(DEGREES): 130
G1(DEGREES)= 130
SELECT G1(DEGREES): 120
G1(DEGREES)= 120
SELECT G1(DEGREES): 110
G1(DEGREES)= 110
SELECT G1(DEGREES): 100
G1(DEGREES)= 100
SELECT G1(DEGREES): 80
G1(DEGREES)= 80
SELECT G1(DEGREES): 70
G1(DEGREES)= 70
SELECT G1(DEGREES): 60
G1(DEGREES)= 60
SELECT G1(DEGREES): 50
G1(DEGREES)= 50
SELECT G1(DEGREES): 40
G1(DEGREES)= 40
SELECT G1(DEGREES): 30
G1(DEGREES)= 30
SELECT G1(DEGREES): 20
G1(DEGREES)= 20
SELECT G1(DEGREES): 10
G1(DEGREES)= 10
```



PROGRAM *LINKAGE3*

This program is essentially the same as *LINKAGE1* except that it has been modified to automatically move the mechanism and display its motion dynamically. Recall that in *LINKAGE1* we were able to produce different configurations of the mechanism by typing in different crank angles. In this program the crank angle is continually increased by the software. The user specifies the start and end angles and the angular increment.

PROGRAM LISTING

Lines 595-597 ask for the start and end crank angles and the increment. These are input in degrees and converted to radians in lines 598-600. The plotting loop starts at line 601 and extends to line 940.

```
10  PRINT "LINKAGE3"
20  PRINT "PLOTS DYNAMIC MOTION"
30  PRINT "OF A 4 BAR LINKAGE"
50  PRINT "READS LINKAGE GEOMETRY"
60  PRINT "FROM DISK"
70  PRINT "TYPE CONT TO CONTINUE"
80  STOP
90  HGR
100  HCOLOR= 7
110  GOSUB 9990
115  DIM P(10),XP(10),YP(10),ZP(10),PC(10)
116  DIM LB(10),LE(10),LC(10)
117  DIM T1(10),T2(10),T3(10),TC(10)
120  INPUT "FILE NAME: ";F$
130  INPUT "DRIVE NUMBER: ";DR
140 D$ = "": REM  CTRL-D
150  PRINT D$;"OPEN";F$;",D";DR
160  PRINT D$;"READ";F$
170  INPUT SF
180  INPUT XO
190  INPUT YO
200  INPUT PX
210  INPUT PY
220  INPUT NP
230  FOR I = 1 TO NP
240  INPUT P(I)
250  INPUT XP(I)
260  INPUT YP(I)
270  INPUT ZP(I)
280  INPUT PC(I)
290  NEXT I
300  INPUT NL
310  FOR I = 1 TO NL
```

LINKAGE3

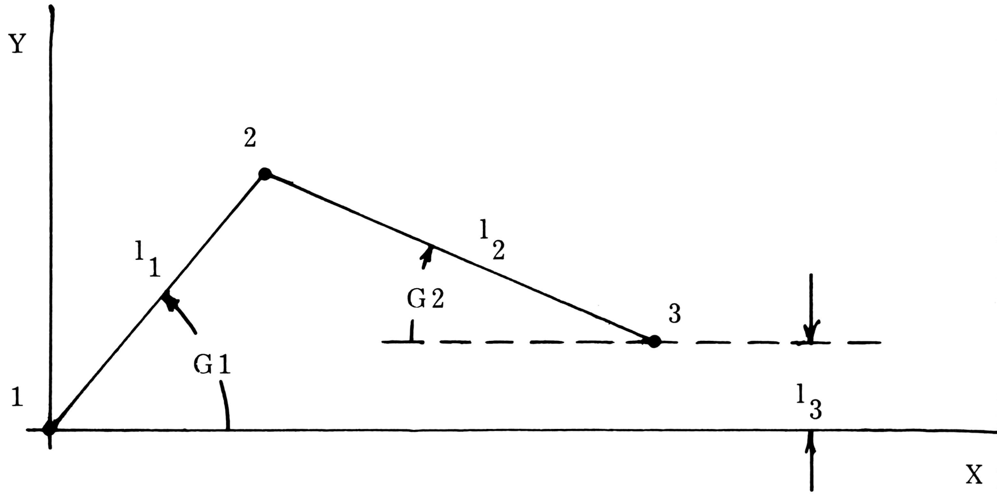
```
  320  INPUT LB(I)
  330  INPUT LE(I)
  340  INPUT LC(I)
  350  NEXT I
  405  PRINT D$;"CLOSE"
  412 X1 = XP(1)
  413 Y1 = YP(1)
  414 X2 = XP(2)
  415 Y2 = YP(2)
  416 X3 = XP(3)
  417 Y3 = YP(3)
  418 X4 = XP(4)
  419 Y4 = YP(4)
  420  GOSUB 730
  430  REM  CALCULATE BAR LENGTHS
  440 A = X2 - X1
  450 B = Y2 - Y1
  460 L1 =  SQR (A * A + B * B)
  470 A = X3 - X2
  480 B = Y3 - Y2
  490 L2 =  SQR (A * A + B * B)
  500 A = X4 - X3
  510 B = Y4 - Y3
  520 L3 =  SQR (A * A + B * B)
  530 A = X4 - X1
  540 B = Y4 - Y1
  550 L0 =  SQR (A * A + B * B)
➤ 595  INPUT "START ANGLE: ";GS
➤ 596  INPUT "END ANGLE: ";GE
➤ 597  INPUT "INCREMENT ANGLE: ";GI
  598 GS = GS * 3.14159 / 180
  599 GE = GE * 3.14159 / 180
  600 GI = GI * 3.14159 / 180
  601  FOR G1 = GS TO GE STEP GI
  611  REM  ERASE LAST PICTURE
  612  HCOLOR= 0
  613  GOSUB 730
  614  HCOLOR= 7
  625  REM  CALCULATE G2
  626 Y2 = L1 *  SIN (G1)
  627 X2 = L1 *  COS (G1)
  628 XX = L0 - (X2 - X1)
  629 YY = Y2
  630 G3 =  ATN (YY / XX)
  640 D =  SQR (XX * XX + YY * YY)
  641  IF D > (L2 + L3) THEN  GOSUB 1000
  650 A = (D * D + L3 * L3 - L2 * L2) / (2 * D * L3)
  651  IF A = 1 THEN  GOSUB 1050
  652  IF A > 1 THEN  GOSUB 2000
  660 G4 =  -  ATN (A /  SQR ( - A * A + 1)) + 1.5708
  670 G2 = G3 + G4
  680  REM  CALCULATE NEW COORDS
  710 X3 = L0 - L3 *  COS (G2)
  720 Y3 = L3 *  SIN (G2)
```

```
725  GOSUB 730
726  NEXT G1
727  GOTO 9998
730  REM  TRANSFORM AND PLOT
740 XR = X1
750 YR = Y1
760  GOSUB 5400
770  HPLOT XT,YT
780 XR = X2
790 YR = Y2
800  GOSUB 5400
810  HPLOT  TO XT,YT
820 XR = X3
830 YR = Y3
840  GOSUB 5400
850  HPLOT  TO XT,YT
860 XR = X4
870 YR = Y4
880  GOSUB 5400
890  HPLOT  TO XT,YT
900 XR = X1
910 YR = Y1
920  GOSUB 5400
930  HPLOT  TO XT,YT
940  RETURN
1000  PRINT "D>(L2+L3)-LINKAGE BOUND"
1010  PRINT "TRY ANOTHER ANGLE"
1020  GOTO 590
1025  RETURN
1050  PRINT "A=1 - LINKAGE BOUND"
1060  PRINT "TRY ANOTHER ANGLE"
1070  GOTO 590
1080  RETURN
2000  PRINT "A>1 - LINKAGE BOUND"
2010  PRINT "TRY ANOTHER ANGLE"
2020  GOTO 590
2030  RETURN
5400  REM  SCALE,ROTATE AND OFFSET PLOT COORDS
5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

LINKAGE4

PROGRAM *LINKAGE4*

This program plots the motion of a slider crank. It is similar in operation and structure to *LINKAGE3* which plotted the motion of a four bar mechanism dynamically.



G1 = crank angle
Point 3 = slider

$$\sin(G2) = (l_1 * \sin(G1) - l_3)/l_2 = A$$

$$\sin^2(G2) + \cos^2(G2) = 1$$

$$\cos(G2) = (1 - \sin^2(G2))^{\frac{1}{2}} = (1 - A*A)^{\frac{1}{2}}$$

$$\tan(G2) = \sin(G2)/\cos(G2) = A/(1 - A*A)^{\frac{1}{2}}$$

PROGRAM LISTING

There are three principal differences between *LINKAGE3* and *LINKAGE4*: First, in the disk reading section, the coordinates of two node points, rather than three, are read in lines 230-290 and

LINKAGE4

only two lines are read in lines 310-350. Second, the sections which calculate the locations of the node points as a function of crank angle (lines 440-655) have been changed to implement the new equations and the plotting routine has been altered accordingly. Third, an addition to this program, which was not incorporated in *LINKAGE3,* is the section from line 671 to 674 which prints the position of the slider vs crank angle on a printer. Line 672 assumes your printer is plugged in to slot 1. If you do not have a printer connected, simply remove these lines.

```
10  PRINT "LINKAGE4"
20  PRINT "PLOTS DYNAMIC MOTION"
30  PRINT "OF A SLIDER-CRANK"
50  PRINT "READS LINKAGE GEOMETRY"
60  PRINT "FROM DISK"
70  PRINT "TYPE CONT TO CONTINUE"
80  STOP
90  HGR
100  HCOLOR= 7
110  GOSUB 9990
115  DIM P(10),XP(10),YP(10),ZP(10),PC(10)
116  DIM LB(10),LE(10),LC(10)
120  INPUT "FILE NAME: ";F$
130  INPUT "DRIVE NUMBER: ";DR
140 D$ = "": REM  CTRL-D
150  PRINT D$;"OPEN";F$;",D";DR
160  PRINT D$;"READ";F$
170  INPUT SF
180  INPUT XO
190  INPUT YO
200  INPUT PX
210  INPUT PY
220  INPUT NP
230  FOR I = 1 TO NP
240  INPUT P(I)
250  INPUT XP(I)
260  INPUT YP(I)
270  INPUT ZP(I)
280  INPUT PC(I)
290  NEXT I
300  INPUT NL
310  FOR I = 1 TO NL
320  INPUT LB(I)
330  INPUT LE(I)
340  INPUT LC(I)
350  NEXT I
405  PRINT D$;"CLOSE"
412 X1 = XP(1)
413 Y1 = YP(1)
414 X2 = XP(2)
415 Y2 = YP(2)
416 X3 = XP(3)
```

```
417 Y3 = YP(3)
420  GOSUB 730
430  REM  CALCULATE BAR LENGTHS
440 A = X2 - X1
450 B = Y2 - Y1
460 L1 =  SQR (A * A + B * B)
470 A = X3 - X2
480 B = Y3 - Y2
490 L2 =  SQR (A * A + B * B)
500 L3 = Y3 - Y1
595  INPUT "START ANGLE: ";GS
596  INPUT "END ANGLE: ";GE
597  INPUT "INCREMENT ANGLE: ";GI
598 GS = GS * 3.14159 / 180
599 GE = GE * 3.14159 / 180
600 GI = GI * 3.14159 / 180
601  FOR G1 = GS TO GE STEP GI
611  REM  ERASE LAST PICTURE
612  HCOLOR= 0
613  GOSUB 730
614  HCOLOR= 7
625  REM  CALCULATE G2
630 A = (L1 *  SIN (G1) - L3) / L2
635  IF A = 1 THEN  GOSUB 1050
636  IF A > 1 THEN  GOSUB 1100
640 G2 =  ATN (A /  SQR ( - A * A + 1))
645 X2 = X1 + L1 *  COS (G1)
650 Y2 = Y1 + L1 *  SIN (G1)
655 X3 = X2 + L2 *  COS (G2)
670  GOSUB 730
671 A = G1 * 180 / 3.14159
672  PRINT D$;"PR#1"
673  PRINT "G1= ";A; TAB( 20);"X3= ";X3
674  PRINT D$;"PR#0"
680  NEXT G1
725  GOTO 9998
726  REM
727  REM
728  REM
729  REM
730  REM  TRANSFORM AND PLOT
740 XR = X1
750 YR = Y1
760  GOSUB 5400
770  HPLOT XT,YT
780 XR = X2
790 YR = Y2
800  GOSUB 5400
810  HPLOT  TO XT,YT
820 XR = X3
830 YR = Y3
840  GOSUB 5400
850  HPLOT  TO XT,YT
```

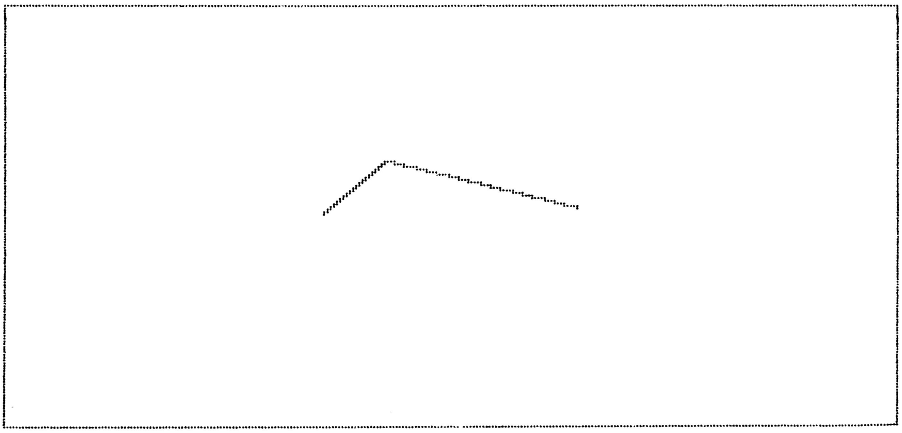
```
940  RETURN
1046  REM
1047  REM
1048  REM
1049  REM
1050  PRINT "A=1 LINKAGE BOUND"
1060  GOTO 9998
1070  RETURN
1096  REM
1097  REM
1098  REM
1099  REM
1100  PRINT "A>1-LINKAGE BOUND"
1110  GOTO 9998
1120  RETURN
5396  REM
5397  REM
5398  REM
5399  REM
5400  REM  SCALE,ROTATE AND OFFSET PLOT COORDS
5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
9986  REM
9987  REM
9988  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

SAMPLE RUN

In the sample run which follows the listing, *CAD20* was loaded and used to create the drawing for the initial configuration of the slider crank. This was stored on disk under the name CAD20SLIDERCRANK. *LINKAGE4* was then loaded and run, recalling drawing CAD20SLIDERCRANK from the disk.

```
CAD20
CREATES A 3D DRAWING
FROM REAL PHYSICAL COORDS
SCALES, ROTATES AND OFFSETS
PLOT COORDINATES
TYPE CONT TO CONTINUE

BREAK IN 40
]CONT
```

```
SF: 10
XO,YO (APPLE UNITS): 100,80
PX: 0
PY: 180
P,L,FILE OR END: P
POINT NO: 1
X,Y,Z COORDS: 0,0,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 2
X,Y,Z COORDS: 2,2,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 3
X,Y,Z COORDS: 8,.2,0
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 1,2
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 2,3
COLOR: 7
P,L,FILE OR END: FILE
FILE NAME: CAD20SLIDERCRANK
DRIVE NUMBER: 1
P,L,FILE OR END: END
ALL DONE
```



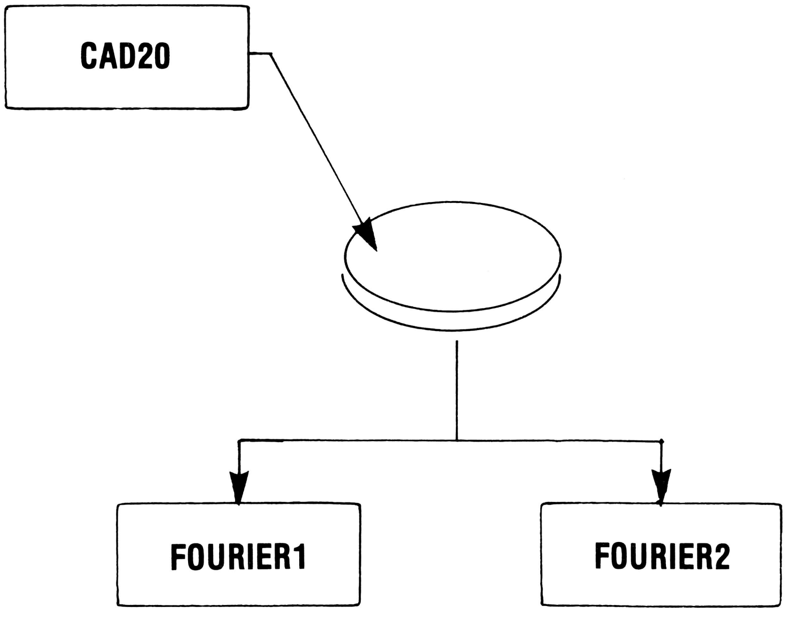
```
]LOAD LINKAGE4
]RUN
LINKAGE4
PLOTS DYNAMIC MOTION
OF A SLIDER-CRANK
READS LINKAGE GEOMETRY
FROM DISK
TYPE CONT TO CONTINUE

BREAK IN 80
]CONT
```

```
FILE NAME: CAD20SLIDERCRANK
DRIVE NUMBER: 1
START ANGLE: 0
END ANGLE: 135
INCREMENT ANGLE: 5
G1= 0              X3= 9.08941747
G1= 5              X3= 9.08167534
G1= 10             X3= 9.04287114
G1= 15             X3= 8.97359906
G1= 20             X3= 8.87485573
G1= 25             X3= 8.74802703
G1= 30             X3= 8.59486979
G1= 35             X3= 8.4174874
G1= 40             X3= 8.2182991
G1= 45             X3= 8.00000173
G1= 50             X3= 7.76552375
G1= 55             X3= 7.51797111
G1= 60             X3= 7.26056515
G1= 65             X3= 6.99657359
G1= 70             X3= 6.72923636
G1= 75             X3= 6.46168932
G1= 80             X3= 6.1968898
G1= 85             X3= 5.93754842
G1= 90             X3= 5.6860719
G1= 95             X3= 5.44452119
G1= 100            X3= 5.21458761
G1= 105            X3= 4.99758806
G1= 110            X3= 4.79447873
G1= 115            X3= 4.60588428
G1= 120            X3= 4.43213875
G1= 125            X3= 4.27333365
G1= 130            X3= 4.12936889
G1= 135            X3= 4.00000279
```

Chapter 4: Fourier Analysis

FOURIER1: Evaluates the first five terms of the Fourier Series for a periodic waveform and plots the magnitudes, *4-1*

FOURIER2: Evaluates the continuous frequency spectrum of the Fourier Transform of a non-periodic waveform and plots the spectrum, *4-8*



PROGRAM *FOURIER1*

In this section we present a program that calculates the magnitudes of the sin and cosine terms in a Fourier series representing a periodic function.

The data describing the periodic function is first created with *CAD20* and stored on disk. The data points do not have to be equally spaced. The function they represent is assumed to vary linearly between data points. FOURIER1 reads the data points from disk, plots the function on the screen, evaluates the terms in the Fourier series, and plots a histogram of the terms on the screen.

ANALYSIS

There are many excellent references on Fourier series and harmonic analysis, in the literature. We will not repeat the basic theory here other than to say that a periodic function f(t) is equivalent to the summation of a series of sin and cosine terms, each of which has the proper amplitude and frequence. The function f does not have to be a function of time; it could just as well be a function of a spatial variable , x, or any other independent variable. We will use the variable t here in a general sense.

The Fourier series is expressed in the following form:

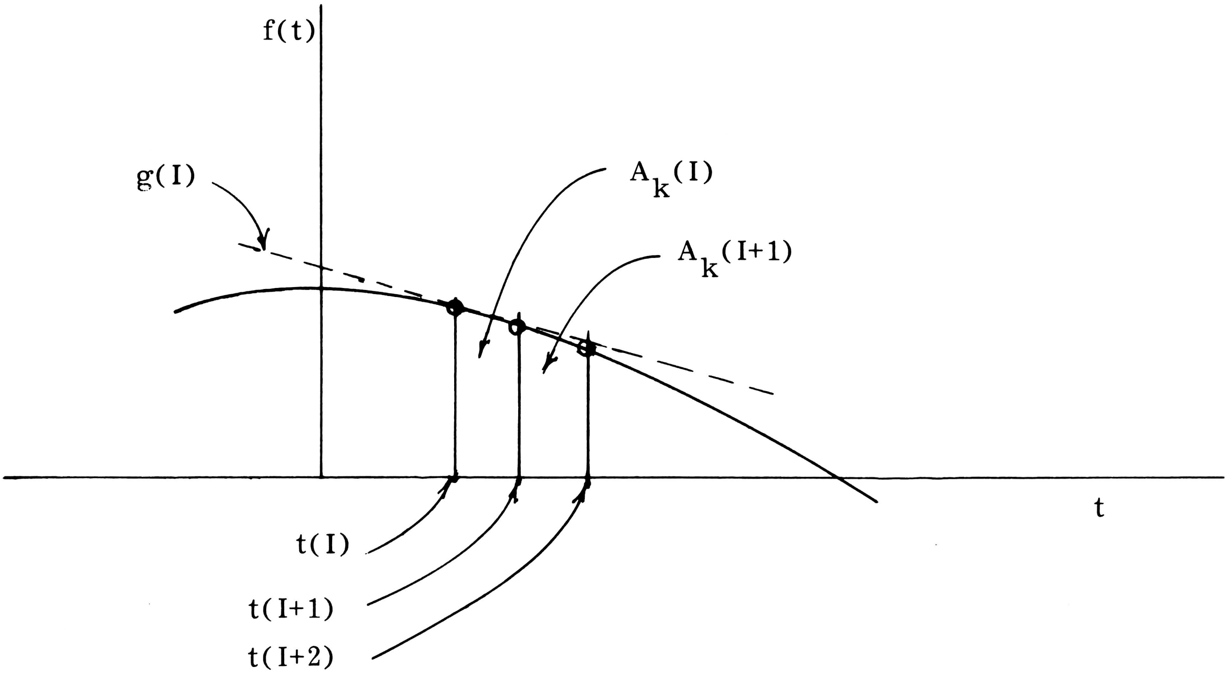
$$f(t) = A_0/2 + \sum_{K=1}^{\infty} (A_k \cos(kwt) + B_k \sin(kwt)$$

$$A_k = (2/T) \int_{-\pi}^{\pi} f(t) \cos(kwt) dt \qquad k = 0,1,2.......$$

$$B_k = (2/T) \int_{-\pi}^{\pi} f(t) \sin(kwt) dt \qquad k = 1,2,3........$$

The function f(t) is repetetive with a base period T. The base frequency, w, is given by:

$$w = 2\pi /T$$


f(t)
g(I)
$A_k(I)$
$A_k(I+1)$
t
t(I)
t(I+1)
t(I+2)

$$A_k(I) = (2/T) \int_{t(I)}^{t(I+1)} g(I) \cos(kwt)dt$$

$$A_k(I+1) = (2/T) \int_{t(I+1)}^{t(I+2)} g(I+1) \cos(kwt)dt$$

$$A_k = \sum_{I=1}^{N-1} A_k(I)$$

Figure 4.1: PIECEWISE LINEAR APPROXIMATION

FOURIER1

The quantity $A_0/2$ is the average value of the function f(t). If f(t) is even (symmetrical on both sides of the f axis), the Fourier series will contain only cosine terms. Conversley, if f(t) is odd (asymmetrical), all A_k's will be zero. The program *FOURIER1* evaluates A_k and B_k.

Figure 4-1 shows the geometry. Data points are first created using *CAD20*. These define f(t) and are stored on disk under a particular drawing name. *FOURIER1* reads them from the disk. Point I 1 is the first data point; I = N is the last. The base period of f(t) is given by:

$$BP = t(N) - t(1)$$

f(t) is assumed to be linear between data points. In reality, this will not always be the case. But if the data points are spaced closely enough, this linear approximation will provide a reasonably good approximation to f(t). The Fourier series is obtained by integrating f(t), a process which tends to smooth errors. This means the Fourier series analysis is insensitive to small errors in f(t) and reasonably good results may be anticipated with this linear approximation.

The reason why we assume a linear variation between data points is that it permits the evaluation of the integrals in the expressions for A_k and B_k to be carried out easily since we can write a closed form expression for the integrals and do not have to resort to a numerical integration scheme in the program. To evaluate the total integral over a period all we have to do is evaluate the difference in value of the closed form solution to the integrals and sum.

This process is illustrated in Figure 4-1. The quantity g(I) is a linear function between two consecutive data points I and I + 1. g(I) is given by:

$$g(I) = SL(I)*t + YI(I)$$

which is the familiar equation for a straight line. SL(I) is the slope and YI(I) is the intercept with the f axis. These quantities are given by:

$$SL(I) = \frac{f(I+1) - f(I)}{t(I+1) - t(I)}$$

$$YI(I) = f(I) - \frac{f(I+1) - f(I)}{t(I+1) - t(I)} \; t(I)$$

Substituting these into g(I) and evaluating the integrals produces the following result:

$$A(k,I) = (2/BP)*(SL(I)*(A1 + A2) + YI(I)*A3)\Big|_{t(I)}^{t(I+1)}$$

$$A1 = (1/k*w)^{2}*\cos(kwt)$$

$$A2 = (1/kw)*t*\sin(kwt)$$

$$A3 = (1/kw)*\sin(kwt)$$

A(k,I) is the contribution to the total integral for A_k of the range between data points at I and I+1. BP is the base period, T.

A similar set of equations can be derived for the B_k terms.

PROGRAM LISTING

Referring to the program listing, the above equations are evaluated in lines 499-690. The loop starts at line 600 and cycles through the first 5 harmonics. (This could be changed to anything desired). The loop starting at line 610 cycles through the data points. The loop from 550-580 evaluates the slope and the intercept of the linear equations over each of the N-1 intervals between data points.

The loop from 710-750 adds up the contribution of the integrals between data points and gives the total integral over the period for each of the five harmonics.

4-5

Fourier Analysis

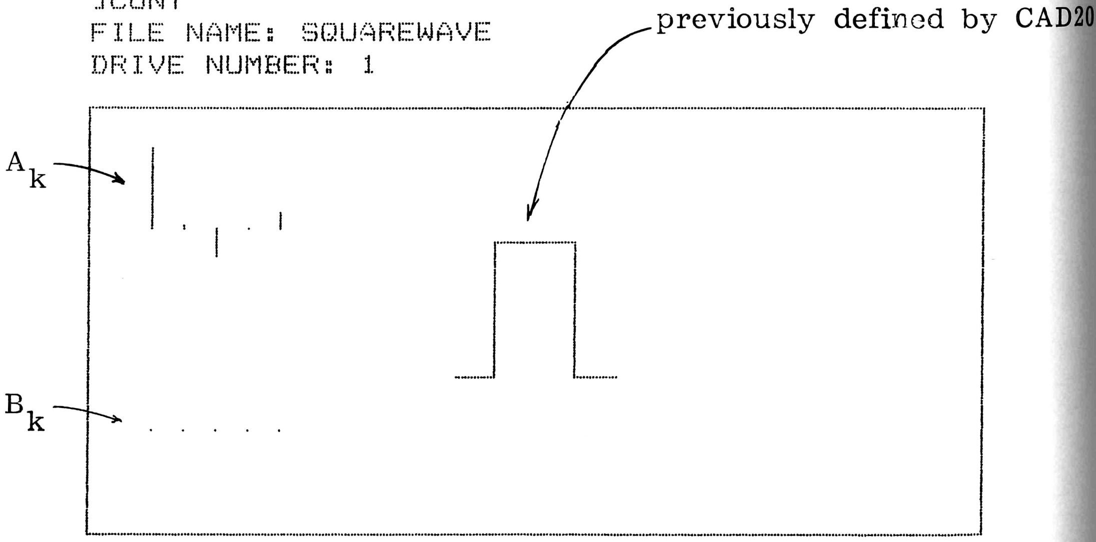
FOURIER1

```
10  PRINT "FOURIER1"
20  PRINT "CALCULATES FOURIER"
30  PRINT "COEFFICIENTS AND PLOTS"
35  PRINT "FREQUENCY SPECTRA"
40  PRINT "TYPE CONT TO CONTINUE"
50  STOP
60  HGR
70  HCOLOR= 7
80  GOSUB 9990
85  DIM P(10),F(10),T(10),ZP(10),PC(10)
86  DIM LB(10),LE(10),LC(10)
87  DIM T1(10),T2(10),T3(10),TC(10)
88  DIM SL(10),YI(10),A(5,10),B(5,10),TA(5),TB(5)
90  INPUT "FILE NAME: ";F$
100  INPUT "DRIVE NUMBER: ";DR
110 D$ = "": REM  CTRL-D
120  PRINT D$;"OPEN";F$;",D";DR
130  PRINT D$;"READ";F$
140  INPUT SF
150  INPUT X0
190  INPUT Y0
200  INPUT PX
210  INPUT PY
220  INPUT NP
230  FOR I = 1 TO NP
240  INPUT P(I)
250  INPUT T(I)
260  INPUT F(I)
270  INPUT ZP(I)
280  INPUT PC(I)
290  NEXT I
300  INPUT NL
310  FOR I = 1 TO NL
320  INPUT LB(I)
330  INPUT LE(I)
340  INPUT LC(I)
350  NEXT I
430  PRINT D$;"CLOSE"
440  GOSUB 3000
499 N = NP
500 BP = T(N) - T(1)
510 BW = 2 * 3.14159 / BP
550  FOR I = 1 TO N - 1
555  IF T(I + 1) = T(I) GOTO 580
560 SL(I) = (F(I + 1) - F(I)) / (T(I + 1) - T(I))
570 YI(I) = F(I) - T(I) * (F(I + 1) - F(I)) / (T(I + 1) - T(I))
580  NEXT I
600  FOR K = 1 TO 5
610  FOR I = 1 TO N - 1
620 AP = K * BW * T(I + 1)
630 AM = K * BW * T(I)
640 A1 = ( COS (AP) -  COS (AM)) / (K * BW * K * BW)
650 A2 = (T(I + 1) *  SIN (AP) - T(I) *  SIN (AM)) / (K * BW)
660 A3 = ( SIN (AP) -  SIN (AM)) / (K * BW)
```

```
670 A(K,I) = (2 / BP) * (SL(I) * (A1 + A2) + YI(I) * A3)
672 A1 = ( SIN (AP) -  SIN (AM)) / (K * BW * K * BW)
673 A2 =  - (T(I + 1) *  COS (AP) - T(I) *  COS (AM)) / (K * BW)
674 A3 =  - ( COS (AP) -  COS (AM)) / (K * BW)
675 B(K,I) = (2 / BP) * (SL(I) * (A1 + A2) + YI(I) * A3)
680  NEXT I
690  NEXT K
700  REM
710  FOR K = 1 TO 5
720  FOR I = 1 TO N - 1
730 TA(K) = TA(K) + A(K,I)
731 TB(K) = TB(K) + B(K,I)
740  NEXT I
750  NEXT K
760  REM  PLOT SPECTRA
770  REM  FIND LARGEST TA(K) OR TB(K)
780 MX = 0
790  FOR K = 1 TO 5
800  IF  ABS (TA(K)) > MX THEN MX =  ABS (TA(K))
810  IF  ABS (TB(K)) > MX THEN MX =  ABS (TB(K))
820  NEXT K
830  REM  SCALE TA(K) AND TB(K) TO 30 PLOT POINTS
840 SC = 30 / MX
850  FOR K = 1 TO 5
860 TA(K) = TA(K) * SC
870 TB(K) = TB(K) * SC
880  NEXT K
890  REM  PLOT A'S AND B'S
900 X1 = 10
910 Y1 = 45
920  FOR K = 1 TO 5
930 X1 = X1 + 10
935 X2 = X1
940 Y2 = Y1 - TA(K)
950  HPLOT X1,Y1 TO X2,Y2
960  NEXT K
970 X1 = 10
980 Y1 = 120
990  FOR K = 1 TO 5
1000 X1 = X1 + 10
1005 X2 = X1
1010 Y2 = Y1 - TB(K)
1020  HPLOT X1,Y1 TO X2,Y2
1030  NEXT K
1040  REM  EVALUATE AVERAGE VALUE, A0
1050 A0 = 0
1060  FOR I = 1 TO N - 1
1070 D = T(I + 1) - T(I)
1080 A0 = A0 + F(I) * D + (F(I + 1) - F(I)) * D / 2
1090  NEXT I
1100 A0 = A0 / BP
1110  PRINT "AVERAGE VALUE= ";A
1120  GOTO 9998
```

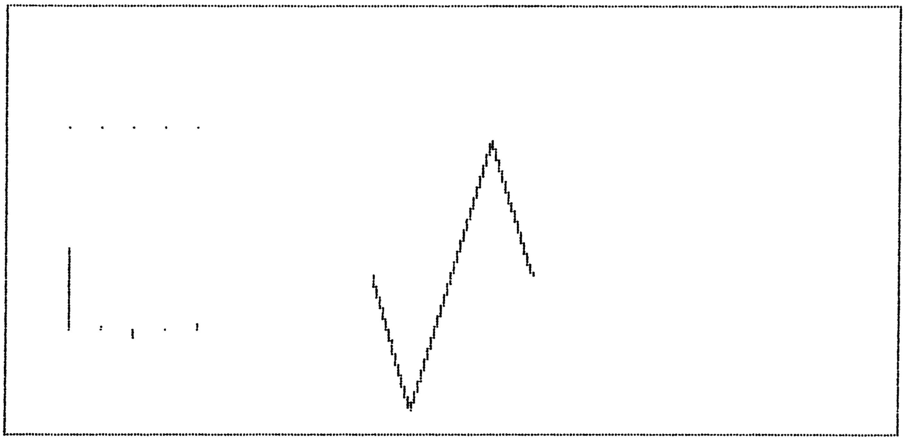
```
3000  REM  RANSFORM AND PLOT INITIAL FUNCTION HERE
3002 XR = T(1)
3003 YR = F(1)
3004  GOSUB 5400
3005  HPLOT XT,YT
3010  FOR I = 2 TO NP
3020 XR = T(I)
3030 YR = F(I)
3040  GOSUB 5400
3050  HPLOT  TO XT,YT
3060  NEXT I
3100  RETURN
5400  REM  SCALE,ROTATE AND OFFSET
5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

```
]RUN
FOURIER1
CALCULATES FOURIER
COEFFICIENTS AND PLOTS
FREQUENCY SPECTRA
TYPE CONT TO CONTINUE

BREAK IN 50
]CONT
FILE NAME: SQUAREWAVE
DRIVE NUMBER: 1
```



previously defined by CAD20

```
]RUN
FOURIER1
CALCULATES FOURIER
COEFFICIENTS AND PLOTS
FREQUENCY SPECTRA
TYPE CONT TO CONTINUE

BREAK IN 50
]CONT
FILE NAME: SAWWAVE
DRIVE NUMBER: 1
```



PROGRAM *FOURIER2*

In program *FOURIER1* we calculated the Fourier series representation of a harmonic function, f(t). That is, we determined the magnitudes of the sin and cosine functions that, when added together, produce f(t). Each term in the series had a discrete frequency, kw_0 where k is a constant and w_0 is the base frequency of f(t). In order for this process to be valid, f(t) must be periodic, or repetitive. The spectrum of components in the Fourier series are separated in frequency by the amount w_0.

In program *FOURIER2* we determine the frequency spectrum of a non-repetitive function, such as a pulse. In this case we might regard the base period as being infinite, implying the base frequency, w_0 is zero. The terms in the series are therefore separated in frequency by zero, producing a continuous spectrum of sin and cosine functions in the series, rather than a discrete spectrum, as is the case with the Fourier series. The result is termed a Fourier Transform. This is usually written in exponential form and you are referred to any standard text on mathematics or signal processing for a thorough discussion. We will summarize only the important features here.

4-9

Fourier Analysis

FOURIER2

ANALYSIS

The following two equations form what is termed a Fourier Transform pair. The first equation defines how the non repetitive function, f(t) can be reconstructed by adding (integrating) an infinite number of sin and cosine terms. This process is analogous to summing the terms in a Fourier series for a repetitive function.

The second equation gives the continuous spectrum F(jw) of the function f(t). This single equation is analogous to the two equations for A_k and B_k in the Fourier series. It is termed the Fourier Transform of f(t).

$$f(t) = (1/2\pi) \int_{-\infty}^{+\infty} F(jw)e^{jwt}dw \qquad F(jw) = \int_{-\infty}^{+\infty} f(t)e^{-jwt}dt$$

where

$$e^{jwt} = \cos(wt) + j\sin(wt)$$

$$e^{-jwt} = \cos(wt) - j\sin(wt)$$

We can combine the last three equations as follows:

$$F(jw) = \int_{-\infty}^{+\infty} f(t)\cos(wt)dt - j\int_{-\infty}^{+\infty} f(t)\sin(wt)dt$$

$$= FA(w) - jFB(w)$$

where

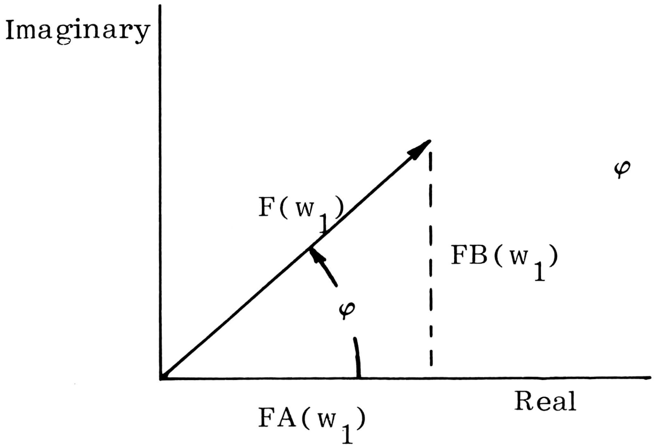
$$FA(w) = \int_{-\infty}^{+\infty} f(t)\cos(wt)dt$$

FOURIER2

$$FB(w) = \int_{-\infty}^{+\infty} f(t)\sin(wt)dt$$

FA(w) is the real component of the spectrum; FB(w) is the imaginary component. The ratio between the two gives the tangent of the phase angle of that particular frequency; the magnitude of the vector sum gives the amplitude.

$$\left|F(w)\right| = \left[FA(w)^2 + FB(w)^2\right]^{1/2}$$



φ = phase angle for frequency w_1 relative to other frequencies in the spectrum

In program *FOURIER2* we will evaluate FA(w) and FB(w) and use them to determine F(w) according to the above equation.

We realize that there is a continuous spectrum of frequencies in the Fourier Transform. We will evaluate FA(w), FB(w) and F(w) at distinct, but close, frequencies. We will them plot a histogram of F(w). This will give an adequate representation of the nature of the continuous spectrum.

PROGRAM LISTING

We will focus our attention on one particular frequency, w_k. $FA(w_k)$ and $FB(w_k)$ are given by:

$$FA(k) = \int_{-\infty}^{+\infty} f(t)\cos(w_k t)dt$$

$$FB(k) = \int_{-\infty}^{+\infty} f(t)\sin(w_k t)dt$$

The magnitude of F at w_k is given by:

$$\left|F(k)\right| = \left[FA(k)^2 + FB(k)^2\right]^{1/2}$$

We can evaluate the above two integrals by following the same procedure that we used in *FOURIER1*. That is, we assume that the function can be adequately represented by linear functions between data points and develop closed form solutions for the integrals which will be valid between data points. In the first of the following equations FA(k,I) is the contribution to the total integral of the span between data points I and I+1 at frequency w_k. Similarly for FB(k,I).

$$FA(k,I) = \int_{-\infty}^{+\infty} g(I)\cos(w_k t)dt$$

$$FB(k\ \ I+1) = \int_{-\infty}^{+\infty} g(I+1)\cos(w_k t)dt$$

As in *FOURIER1* g(I) is a linear function between data points I and I + 1. g(I) is given by:

$$g(I) = SL(I)*t + YI(I)$$

$$SL(I) = \frac{f(I+1) - f(I)}{t(I+1) - t(I)}$$

$$YI(I) = \frac{f(I) - f(I+1) - f(I)}{t(I+1) - t(I)} * t(I)$$

Substituting the above into the integrals and integrating gives the following:

$$FA(k,I) = SL(I)*(A1 + A2) + YI(I)*A3 \Big|_{t(I)}^{t(I+1)}$$

$$A1 = (1/w_k)^2*\cos(w_k t)$$

$$A2 = (1/w_k)*t*\sin(w_k t)$$

$$A3 = (1/w_k)*\sin(w_k t)$$

Similarly for FB(k,I).

As previously mentioned *FOURIER2* is practically identical to *FOURIER1* in structure. The main difference is that *FOURIER1* evaluated the integrals at 5 discrete frequencies. *FOURIER2* evaluates them over a range of frequencies from w_{max} to w_{min} in increments dw which are specified by the user in lines 600-602. Loop 604-690 carries out the integration at these frequencies.

All equations in the loop have been modified to incorporate the frequency w_k rather than $k*B_w$. TA(k) and TB(k) in lines 670 and 675 are the total integrals involved in the expressions for FA(k) and FB(k). TT(k) in line 678 is the quantity F(k), the total vectorial magnitude of the spectrum at frequency w_k.

If the frequency w_k is zero, we run into trouble since several expressions in the loop involve division by w_k which will produce a "division by zero" error message. So, line 606 checks to see if the frequency is zero. If it is, control drops to subroutine 2000 where that component is evaluated. Note that it is equal to the area under the function f(t).

This program also incorporates a routine which will print components of the spectrum on a printer. This is done in subroutine 6000

```
10  PRINT "FOURIER2"
20  PRINT "CALCULATES FOURIER"
30  PRINT "TRANSFORM AND PLOTS"
35  PRINT "FREQUENCY SPECTRA"
40  PRINT "TYPE CONT TO CONTINUE"
50  STOP
60  HGR
70  HCOLOR= 7
80  GOSUB 9990
85  DIM P(20),F(20),T(20),ZP(20),PC(20)
86  DIM LB(20),LE(20),LC(20)
87  DIM TA(20),TB(20),TT(20),W(20)
88  DIM SL(20),YI(20)
90  INPUT "FILE NAME: ";F$
100  INPUT "DRIVE NUMBER: ";DR
110 D$ = "": REM  CTRL-D
120  PRINT D$;"OPEN";F$;",D";DR
130  PRINT D$;"READ";F$
140  INPUT SF
150  INPUT XO
190  INPUT YO
200  INPUT PX
210  INPUT PY
220  INPUT NP
230  FOR I = 1 TO NP
240  INPUT P(I)
250  INPUT T(I)
```

```
260  INPUT F(I)
270  INPUT ZP(I)
280  INPUT PC(I)
290  NEXT I
300  INPUT NL
310  FOR I = 1 TO NL
320  INPUT LB(I)
330  INPUT LE(I)
340  INPUT LC(I)
350  NEXT I
430  PRINT D$;"CLOSE"
440  GOSUB 3000
499 N = NP
550  FOR I = 1 TO N - 1
555  IF T(I + 1) = T(I) GOTO 580
560 SL(I) = (F(I + 1) - F(I)) / (T(I + 1) - T(I))
570 YI(I) = F(I) - T(I) * (F(I + 1) - F(I)) / (T(I + 1) - T(I))
580  NEXT I
600  INPUT "WMAX: ";WX
601  INPUT "WMIN: ";WM
602  INPUT "DW: ";DW
603 K = 0
604  FOR WK = WM TO WX STEP DW
605 K = K + 1
606  IF WK = 0 THEN  GOSUB 2000
608 W(K) = WK
609  IF WK = 0 THEN  GOTO 690
615  FOR I = 1 TO NL
620 AP = WK * T(I + 1)
630 AM = WK * T(I)
640 A1 = ( COS (AP) -  COS (AM)) / (WK * WK)
650 A2 = (T(I + 1) *  SIN (AP) - T(I) *  SIN (AM)) / WK
660 A3 = ( SIN (AP) -  SIN (AM)) / WK
➤ 670 TA(K) = TA(K) + SL(I) * (A1 + A2) + YI(I) * A3
672 A1 = ( SIN (AP) -  SIN (AM)) / (WK * WK)
673 A2 =  - (T(I + 1) *  COS (AP) - T(I) *  COS (AM)) / WK
674 A3 =  - ( COS (AP) -  COS (AM)) / WK
➤ 675 TB(K) = TB(K) + SL(I) * (A1 + A2) + YI(I) * A3
➤ 678 TT(K) =  SQR (TA(K) * TA(K) + TB(K) * TB(K))
680  NEXT I
685  PRINT "WK= ";WK; TAB( 20);"F(WK)= ";TT(K)
690  NEXT WK
695 KK = K
700  REM
760  REM  PLOT SPECTRA
770  REM  FIND LARGEST TT(K)
780 MX = 0
790  FOR K = 1 TO KK
800  IF  ABS (TT(K)) > MX THEN MX =  ABS (TT(K))
820  NEXT K
830  REM  SCALE TT(K) TO 30 PLOT UNITS
831  REM  SCALE W TO 40 PLOT UNITS
840 SC = 30 / MX
846 SW = 40 / (WX - WM)
```

```
850  FOR K = 1 TO KK
860 TT(K) = TT(K) * SC
875 W(K) = W(K) * SW
880  NEXT K
890  REM  PLOT T'S
920  FOR K = 1 TO KK
930 X1 = 10 + W(K)
935 X2 = X1
940 Y2 = 80 - TT(K)
945 Y1 = 80
950  HPLOT X1,Y1 TO X2,Y2
960  NEXT K
1120  GOTO 6000
2000  REM  CALCULATE WK=0 COMPONENT
2001 A0 = 0
2020  FOR I = 1 TO N - 1
2030 D = T(I + 1) - T(I)
2040 A0 = A0 + F(I) * D + (F(I + 1) - F(I)) * D / 2
2050  NEXT I
2085 TT(K) = A0
2086  PRINT "WK= ";WK; TAB( 20);"F(WK)= ";TT(K)
2090  RETURN
3000  REM  TRANSFORM AND PLOT INITIAL FUNCTION
3002 XR = T(1)
3003 YR = F(1)
3004  GOSUB 5400
3005  HPLOT XT,YT
3010  FOR I = 2 TO NP
3020 XR = T(I)
3030 YR = F(I)
3040  GOSUB 5400
3050  HPLOT  TO XT,YT
3060  NEXT I
3100  RETURN
5400  REM  SCALE,ROTATE AND OFFSET
5410 XT = X0 + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = Y0 + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
6000  REM  PRINT RESULTS
6010  INPUT "WANT TO PRINT? (Y/N): ";CK$
6020  IF CK$ = "N" GOTO 9998
6030  IF CK$ = "Y" GOTO 6040
6035  GOTO 6010
6040  PRINT D$;"PR#1"
6042  PRINT
6043  PRINT "SPECTRUM FOR DWG ";F$
6044  PRINT
6045  PRINT "FREQUENCY"; TAB( 30);"AMPLITUDE"
6046  PRINT
6049 K = 0
6050  FOR WK = WM TO WX STEP DW
6051 K = K + 1
6055 TT(K) = TT(K) / SC
```

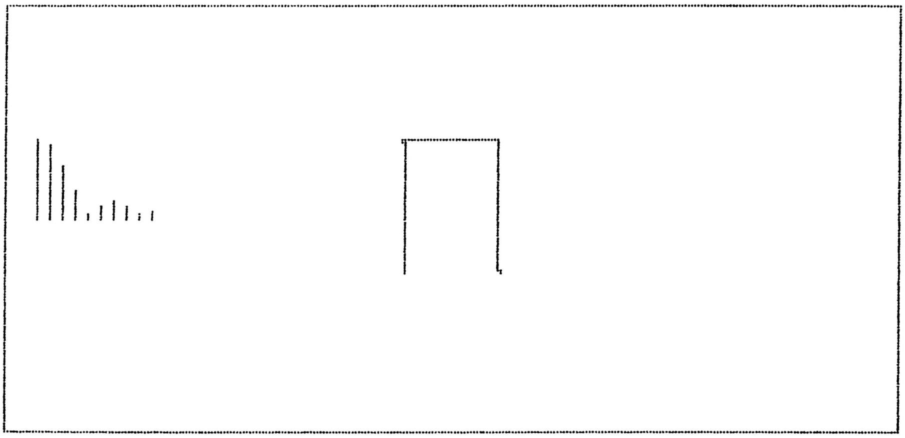
SAMPLE RUN 1

In this sample run, *CAD20* was used to create and store a pulse. It was stored on disk under the name CAD20PULSE. *FOURIER2* was then loaded and run. CAD20PULSE was recalled from disk and the Fourier Transform carried out.

In the figure that follows, the bars to the left of the pulse indicate the relative magnitude of the components in the spectrum. In the listing of amplitude vs frequency, note that F(0) is equal to the area under the waveform in real units, not Apple plotting units. This is because the waveform was created and stored on disk in real physical units. These are the units that were operated on by *FOURIER2*. Plotting units were used only when plotting was done. It can be seen that results agree well with closed form solutions.

```
]LOAD CAD20
]RUN
CAD20
CREATES A 3D DRAWING
FROM REAL PHYSICAL COORDS
SCALES, ROTATES AND OFFSETS
PLOT COORDINATES
TYPE CONT TO CONTINUE

BREAK IN 40
]CONT
SF: 1
XO,YO (APPLE UNITS): 140,100
PX: 0
PY: 180
P,L,FILE OR END: P
POINT NO: 1
X,Y,Z COORDS: -15,0,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 2
X,Y,Z COORDS: -15,50,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 3
X,Y,Z COORDS: 15,50,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 4
X,Y,Z COORDS: 15,0,0
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 1,2
```

```
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 2,3
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 3,4
COLOR: 7
P,L,FILE OR END: FILE
FILE NAME: CAD20PULSE
DRIVE NUMBER: 1

]RUN
FOURIER2
CALCULATES FOURIER
TRANSFORM AND PLOTS
FREQUENCY SPECTRA
TYPE CONT TO CONTINUE

BREAK IN 50
]CONT
FILE NAME: CAD20PULSE
DRIVE NUMBER: 1
```



```
SPECTRUM FOR DWG CAD20PULSE

FREQUENCY                       AMPLITUDE

WK= 0                           F(WK)= 1500
WK= .05                         F(WK)= 1363.27752
WK= .1                          F(WK)= 997.494988
WK= .15                         F(WK)= 518.715465
WK= .2                          F(WK)= 70.5600039
WK= .25                         F(WK)= 228.624528
WK= .3                          F(WK)= 325.843373
WK= .35                         F(WK)= 245.409855
WK= .4                          F(WK)= 69.853874
WK= .45                         F(WK)= 100.009795
```

4-18 SAMPLE RUN 2

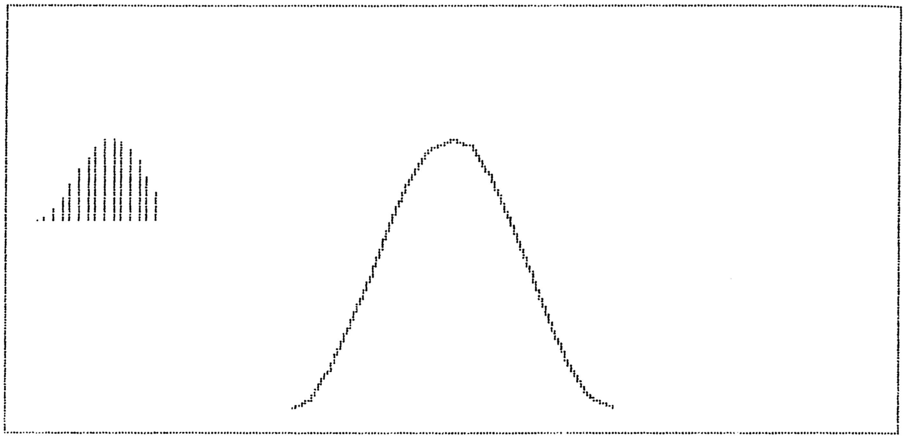
In this run, we use *CAD20* to create and store a cosine function on disk. *FOURIER2* reads the data describing the cosine function from the disk and carries out the transformation.

```
]LOAD CAD20,D2
]RUN
CAD20
CREATES A 3D DRAWING
FROM REAL PHYSICAL COORDS
SCALES, ROTATES AND OFFSETS
PLOT COORDINATES
TYPE CONT TO CONTINUE

BREAK IN 40
]CONT
SF: 50
X0,Y0 (APPLE UNITS): 140,100
PX: 0
PY: 180
P,L,FILE OR END: P
POINT NO: 1
X,Y,Z COORDS: -1,-1,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 2
X,Y,Z COORDS: -.875,-.924,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 3
X,Y,Z COORDS: -.75,-.707,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 4
X,Y,Z COORDS: -.625,-.383,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 5
X,Y,Z COORDS: -.5,0,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 6
X,Y,Z COORDS: -.375,.383,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 7
X,Y,Z COORDS: -.25,.707,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 8
X,Y,Z COORDS: -.125,.924,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 9
X,Y,Z COORDS: 0,1,0
COLOR: 7
```

```
P,L,FILE OR END: P
POINT NO: 10
X,Y,Z COORDS: .125,.924,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 11
X,Y,Z COORDS: .25,.707,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 12
X,Y,Z COORDS: .375,.383,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 13
X,Y,Z COORDS: .5,0,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 14
X,Y,Z COORDS: .625,-.383,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 15
X,Y,Z COORDS: .75,-.707,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 16
X,Y,Z COORDS: .875,-.924,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 17
X,Y,Z COORDS: 1,-1,0
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 1,2
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 2,3
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 3,4
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 4,5
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 5,6
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 6,7
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 7,8
COLOR: 7
```

```
P,L,FILE OR END: L
BEGIN/END PTS 8,9
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 9,10
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 10,11
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 11,12
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 12,13
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 13,14
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 14,15
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 15,16
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 16,17
COLOR: 7
P,L,FILE OR END: FILE
FILE NAME: CAD20COSINE2
DRIVE NUMBER: 1
P,L,FILE OR END: END
ALL DONE

]RUN
FOURIER2
CALCULATES FOURIER
TRANSFORM AND PLOTS
FREQUENCY SPECTRA
TYPE CONT TO CONTINUE

BREAK IN 50
]CONT
FILE NAME: CAD20COSINE2
DRIVE NUMBER: 1

SPECTRUM FOR DWG CAD20COSINE2

FREQUENCY                    AMPLITUDE

WK= 0                        F(WK)= -2.91038305E-11
WK= .4                       F(WK)= .0316783618
WK= .8                       F(WK)= .122780367
WK= 1.2                      F(WK)= .26199584
WK= 1.6                      F(WK)= .432041721
```

```
WK= 2                                    F(WK)= .611800903
WK= 2.4                                  F(WK)= .778921506
WK= 2.8                                  F(WK)= .912545023
WK= 3.2                                  F(WK)= .995818011
WK= 3.6                                  F(WK)= 1.01787348
WK= 4                                    F(WK)= .975040209
WK= 4.40000001                           F(WK)= .871140901
WK= 4.80000001                           F(WK)= .716859724
WK= 5.2                                  F(WK)= .528279605
WK= 5.6                                  F(WK)= .324794672
```



SAMPLE RUN 3

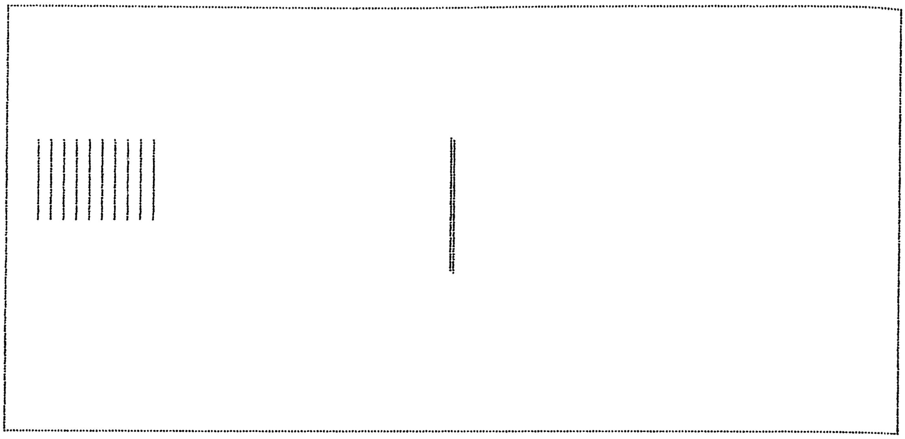
In this last sample run, we find the Fourier Transform of an impulse function. An ideal impulse has an infinite amplitude and a zero duration. It is, of course, impossible to produce an ideal impulse in a computer since we can't represent an infinitely large number. However, this limitation isn't as serious as it may seem since ideal impulses don't occur in nature either. So, we will take the Fourier Transform of a non-ideal impulse - a pulse which has a very large amplitude and a very short duration.

As you may note from the printout of F(w) vs w_k, the zero frequency term gives the area under the impulse which is commonly termed the "strength" of the impluse.

```
]RUN
CAD20
CREATES A 3D DRAWING
FROM REAL PHYSICAL COORDS
SCALES, ROTATES AND OFFSETS
PLOT COORDINATES
TYPE CONT TO CONTINUE

BREAK IN 40
]CONT
```

```
SF: .00005
XO,YO (APPLE UNITS): 140,100
PX: 0
PY: 180
P,L,FILE OR END: P
POINT NO: 1
X,Y,Z COORDS: -.000001,0,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 2
X,Y,Z COORDS: -.000001,1000000,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 3
X,Y,Z COORDS: .000001,1000000,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 4
X,Y,Z COORDS: .000001,0,0
COLOR: 7
P,L,FILE OR END: L
BEGIN/END PTS 1,2


]LOAD FOURIER2,D1
]RUN
FOURIER2
CALCULATES FOURIER
TRANSFORM AND PLOTS
FREQUENCY SPECTRA
TYPE CONT TO CONTINUE

BREAK IN 50
]CONT
FILE NAME: CAD20IMPULSE
DRIVE NUMBER: 1
WMAX: .5
WMIN: 0
DW: .05


SPECTRUM FOR DWG CAD20IMPULSE

FREQUENCY                    AMPLITUDE

WK= 0                        F(WK)= 2
WK= .05                      F(WK)= 1.99979759
WK= .1                       F(WK)= 1.99991188
WK= .15                      F(WK)= 1.99994997
WK= .2                       F(WK)= 1.99998331
WK= .25                      F(WK)= 1.99999188
WK= .3                       F(WK)= 1.9999595
WK= .35                      F(WK)= 1.99996902
```

```
WK= .4                                  F(WK)= 1.99998331
WK= .45                                 F(WK)= 1.99998807
```

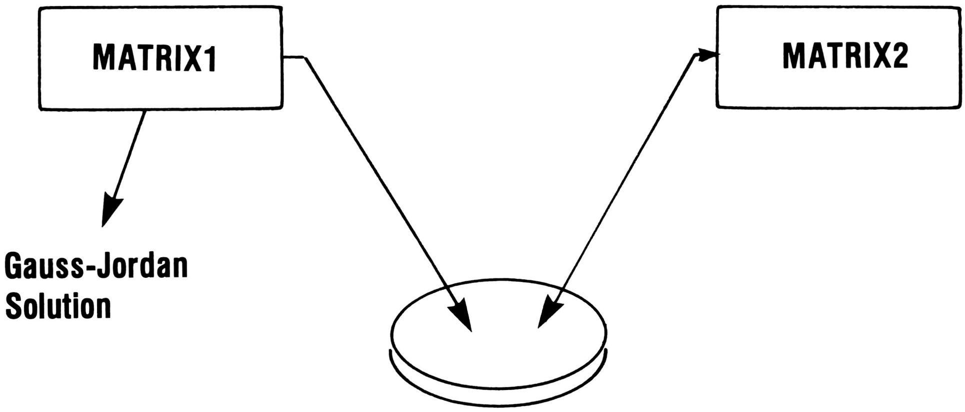


Chapter 5: Matrix Methods

Gauss-Jordan Elimination, *5-1*
- **MATRIX1**: Solves a set of simultaneous linear equations using the Gauss-Jordan Elimination technique, *5-9*

Thermal Analysis, *5-15*

Structural Analysis, *5-20*
- **MATRIX2**: Recalls a matrix from disk, updates it and replaces it on disk, *5-24*


MATRIX1
MATRIX2
Gauss-Jordan
Solution

5-1 GAUSS-JORDAN ELIMINATION

Matrix Methods

There are a wide range of problems that can be reduced to a set of simultaneous linear equations. The solution to the problem then becomes one of solving the equations simultaneously

GAUSS-JORDAN

In this chapter, we develop a program which solves simultaneous equations using the Gauss-Jordan elimination technique. Gauss-Jordan is termed a "direct" method of solution since it gives us a solution to the equations directly, without having to invert a matrix. There are several other direct methods of solution but the Gauss-Jordan method is generally recognized as being one of the most efficient. It is also one of the easiest to understand and program.

We will illustrate the technique by an example. Consider the electric circuit shown in Figure 5-1. Branch currents i1, i2 and i3 are assumed to flow in the directions shown. The known variables are E1, E2, R1, R2 and R3. The unknowns are the currents i1, i2 and i3.

Since there are three unknowns, we need three equations to solve for them. We can get one of the three by summing currents flowing into node d and setting the result equal to zero (nodes don't accumulate current).

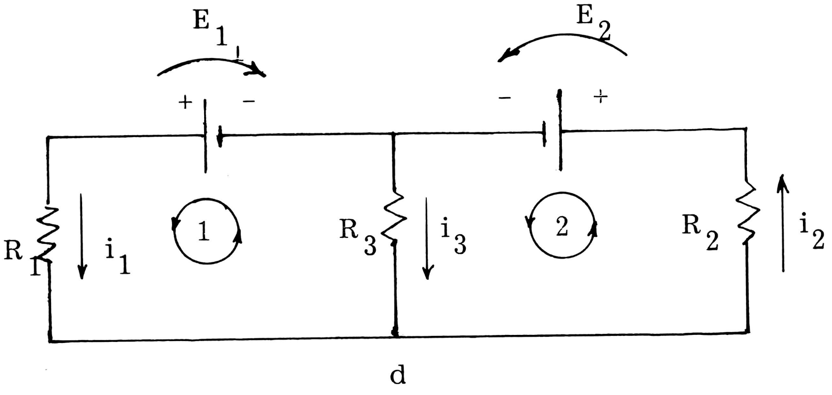


Figure 5.1: Electric Circuit

$$i1 + i3 - i2 = 0$$

We can get two more equations by summing voltages in each loop and equating to zero:

$$-R_1*i_1 + R_3*i_3 + E_1 = 0$$

$$-R_3*i_3 - R_2*i_2 - E_2 = 0$$

The convention used in deriving the above two equations is that when an element in a loop is "crossed", its voltage is negative (drops) when travelling with the current and is positive (rises) when traveling against the current, somewhat like swimming upstream and meeting a waterfall. The loop equations above were derived by travelling around each loop in the counter-clockwise direction.

We can re-arrange these three equations as follows:

$$i_1 - i_2 + i_3 = 0$$

$$-R_1*i_1 + (0)*i_2 + R_3*i_3 = -E_1$$

$$(0)*i_1 - R_2*i_2 - R_3*i_3 = E_2$$

In matrix notation:

$$\begin{bmatrix} 1 & -1 & 1 \\ -R_1 & 0 & R_3 \\ 0 & -R_2 & -R_3 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -E_1 \\ E_2 \end{bmatrix}$$

$$\begin{bmatrix} R \end{bmatrix} \begin{bmatrix} i \end{bmatrix} = \begin{bmatrix} E \end{bmatrix}$$

GAUSS JORDAN

The coefficients in the matrix of resistances, R, are defined as follows where J indicates the columns and I the rows:

$R(1,1) = 1 \qquad R(1,2) = -1 \qquad R(1,3) = 1$

$R(2,1) = -R_1 \qquad R(2,2) = 0 \qquad \ldots\ldots\ldots$

$\ldots\ldots\ldots \qquad \ldots\ldots\ldots\ldots \qquad \ldots\ldots\ldots\ldots$

Let's assign values to the R's and E's and solve for the unknown currents using the Gauss-Jordan method. We'll use the following values:

R1 = 10
R2 = 20
R3 = 30
E1 = 50
E2 = 20

Substituting these into the matrix:

$$\begin{array}{l} \text{row (1)} \\ \\ \text{row (2)} \\ \\ \text{row (3)} \end{array} \quad \begin{bmatrix} 1 & -1 & 1 \\ \\ -10 & 0 & 30 \\ \\ 0 & -20 & -30 \end{bmatrix} \begin{bmatrix} i_1 \\ \\ i_2 \\ \\ i_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \\ -50 \\ \\ 20 \end{bmatrix}$$

We will now solve these equations using the Guass-Jordan technique, and we'll do it the hard way - by hand - to illustrate the method. This will make it easier to develop and understand the software.

The general idea is to get 1's and 0's in the R matrix by multiplying various rows by constants and then subtracting them from other rows. By a row we mean the row of R's plus the associated E since, if we are to multiply one side of the equation by a constant, we must multiply the other side as well, and a row is just an equation.

We'll work our way across the R matrix from left to right "reducing" rows to end up with 1's and 0's. The solution will then be obvious.

Starting at column 1, find the row containing the largest (absolute value) number. This is row 2 with the -10 in column 1. We choose the largest number since this eliminates the chance of using a zero, which we don't want to do since we will be dividing by this number later. Also, the larger the number we choose at this point, the greater will be the accuracy of our calculations.

To get a zero at R(1,1), write a dummy row by dividing row 2 by -10:

$$\begin{matrix} +1 & 0 & -3 & +5 \end{matrix}$$

Now multiply this dummy row by -1 [ie: -R(1,1)]:

$$\begin{matrix} -1 & 0 & +3 & -5 \end{matrix}$$

and add to row 1:

$$\begin{matrix} 0 & -1 & 4 & -5 \end{matrix}$$

This is the new form of row 1. The set of equations becomes:

$$\begin{bmatrix} 0 & -1 & 4 \\ 1 & 0 & -3 \\ 0 & -20 & -30 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} -5 \\ +5 \\ 20 \end{bmatrix}$$

We could do the same for row 3 to get a zero in position R(1,2), but there is already a zero there.

Now, we move to column 2 and repeat. Here the pivot is -20. First, we will reduce row 1 to get a zero at point R(1,2). Divide row 3 by -20 to get the dummy row:

$$\begin{matrix} 0 & 1 & 1.5 & -1 \end{matrix}$$

Multiply by +1 [ie: -(-1)] to reduce row 1:

$$\begin{matrix} 0 & 1 & 1.5 & -1 \end{matrix}$$

and add to row 1:

$$\begin{bmatrix} 0 & 0 & 5.5 \\ 1 & 0 & -3 \\ 0 & 1 & 1.5 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} -6 \\ 5 \\ -1 \end{bmatrix}$$

We could do the same for row 2 using the dummy row, but there is already a zero at point R(2,2).

Now, go to column 3 and repeat. The pivot row is row 1 since this is where the largest constant is.

What would happen if the largest constant was in some other row, say row 2, that had already been used as a pivot row? We couldn't use it because of the 1 to the left of the pivot column. There will always be a 1 to the left of the pivot column in a row that has already been used as a pivot row. When we go to add the dummy row to another row, this term will replace a zero, which we want, with a constant. We must therefore use the row with the largest constant as the pivot row, but it must not have been previously used as a pivot row. In this present case, row 1 must be used and, fortunately, it also contains the largest constant in column 3.

We get the dummy row by dividing row 1 by 5.5 .

$$\begin{matrix} 0 & 0 & 1 & -1.09 \end{matrix}$$

To reduce row 2, multiply this by 3:

$$\begin{matrix} 0 & 0 & 3 & -3.27 \end{matrix}$$

and add to row 2:

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1.5 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} -1.09 \\ 1.73 \\ -1 \end{bmatrix}$$

To reduce row 3, multiply the dummy row by -1.5:

$$\begin{matrix} 0 & 0 & -1.5 & 1.64 \end{matrix}$$

and add to row 3:

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} -1.09 \\ 1.73 \\ .64 \end{bmatrix}$$

Phew! Glad that's over. Now, the solution is obvious.

$$(0)*i_1 + (0)*i_2 + (1)*i_3 = -1.03 \qquad i_3 = -1.09$$

$$(1)*i_1 + (0)*i_2 + (0)*i_3 = 1.73 \qquad i_1 = 1.73$$

$$(0)*i_1 + (1)*i_2 + (0)*i_3 = .64 \qquad i_2 = .64$$

SAMPLE RUN

Before explaining how program *MATRIX1* carries out this process, we will use it to solve this same problem:

```
]RUN
MATRIX1
SOLVES A SET OF
LINEAR EQUATIONS BY
THE GAUSS-JORDAN TECHNIQUE
TYPE CONT TO CONTINUE

BREAK IN 55
]CONT
WANT TO READ MATRIX FROM DISK? (Y/N): N
NUMBER OF EQUATIONS: 3

   EQUATION 1

   COEFFICIENT 1
   A(1,1):
?1

   COEFFICIENT 2
   A(1,2):
?-1

   COEFFICIENT 3
   A(1,3):
?1

   CONSTANT 4
   C(1):
?0


   EQUATION 2
```

```
     COEFFICIENT 1
     A(2,1):
?-10

     COEFFICIENT 2
     A(2,2):
?0

     COEFFICIENT 3
     A(2,3):
?30


     CONSTANT 4
     C(2):
?-50


     EQUATION 3

     COEFFICIENT 1
     A(3,1):
?0

     COEFFICIENT 2
     A(3,2):
?-20

     COEFFICIENT 3
     A(3,3):
?-30

     CONSTANT 4
     C(3):
?20

WANT TO SAVE DATA ON DISK? (Y/N): Y
FILE NAME: MATRIX1CIRCUIT
DRIVE NUMBER: 1
WANT TO PRINT MATRIX? (Y/N): Y


INPUT MATRIX


ROW 1
        A(1,1)= 1
        A(1,2)= -1
        A(1,3)= 1
        A(1,4)= 0
```

```
ROW  2
          A(2,1)=  -10
          A(2,2)=  0
          A(2,3)=  30
          A(2,4)=  -50

ROW  3
          A(3,1)=  0
          A(3,2)=  -20
          A(3,3)=  -30
          A(3,4)=  20



RESULTS

    X(1)=  1.72727273  ←  i1

    X(2)=  .636363636  ←  i2

    X(3)=  -1.09090909  ←  i3
```

These results are the same as we obtained doing it by hand. These are more accurate, however, since the hand calculations were carried out on a pocket calculator.

PROGRAM LISTING

Program *MATRIX1* solves a set of linear equations using the Gauss-Jordan elimination technique. The basic method was explained in detail in the preceding section. Here, we present the program listing and point out where the various operations take place.

In line 60, the user has the option of reading a data set from the disk or creating one through the keyboard. If the option to type one in through the keyboard is selected, the data may be written to disk for later use.

If the user types "N", (option to enter through keyboard is selected), control drops to line 80. The numbers to be typed in will be placed in the A(I,J) matrix. I indicates the row; J the column. Line 120 asks for the number of equations. This is IT, the total number of rows. Note that JT is IT + 1. The matrix of coefficients is square, but we are increasing the number of columns by one here to accommodate the constant on the right side of the equations. That is, a typical equation would be:

$$A(I,1)*y_1 + A(I,2)*y_2 +......+ A(I,JT-1)*y(JT-1) = A(I,JT)$$

The program asks for the A's for NT such equations. Each equation will have NT + 1 A's.

After typing in the matrix, the user may select to store the data file on disk. This is accomplished in subroutine 2000. He is then given the option of having the matrix printed out on a printer. This is done in subroutine 5000.

Next, starting at line 260, the program begins the Gauss-Jordan reduction. Line 300 transfers control to subroutine 3000 which finds the maximum value of the pivot column. The program starts at column 1 and advances to the right. This is done by the loop which extends from line 295 to 500.

After the pivot row is found, the size of the pivot number is checked. If it is too small there will not be sufficient accuracy in the solution so the program stops. Reduction of the pivot row takes place in loop 400-420. The other rows are reduced, one at a time, starting just to the right of the pivot column in loop 400-490. Answers are stored in the X(I) array in loop 520-550. Results are then printed.

Subroutine 2500 checks the results of the computation. It does this by recalling the original matrix from disk. It was necessary to store the original data since the computations destroyed the original values. The matrix is multiplied by the solution vector to give the loadings corresponding to the solution. These are the check values, C(k). These are compared with the original loads which were input as part of the A(I,J) array. If the solution was perfect, these two quantities should be the same. The difference between the two is a measure of the relative accuracy of the solution.

A few words should be said about subroutine 3000 which finds the pivot rows. L(I) was originally set equal to the row numbers in loop 270-280. If a row is used as a pivot row, the corresponding L(I) is set equal to zero. Line 3050 checks to see if L(I) = 0 for a particular row. If it does, then it has been previously used as a pivot row and subroutine 3000 won't use it again. Line 3040 sets Y(I) equal to the coefficients in column J, the pivot column. Loop 3080-3100 finds the maximum value, which indicates the pivot row,

M. Note that M will be used later in line 540 when it comes time to determine which variables are which in the solution. The 1's are scattered all over the matrix; M keeps track of them.

MATRIX1

```
   5  HOME
   10  PRINT "MATRIX1"
   20  PRINT "SOLVES A SET OF"
   30  PRINT "LINEAR EQUATIONS BY"
   40  PRINT "THE GAUSS-JORDAN TECHNIQUE"
   50  PRINT "TYPE CONT TO CONTINUE"
   55  STOP
   56  DIM A(20,20),K(20),L(20),C(20),X(20)
   57  DIM Y(20)
➤  60  INPUT "WANT TO READ MATRIX FROM DISK? (Y/N): ";CK$
   65  IF CK$ = "N" THEN  GOTO 80
   66  IF CK$ = "Y" THEN  GOTO 69
   68  GOTO 60
   69  REM
   70 XX = 1
   71  GOSUB 2500
   72  GOTO 256
   80  REM
➤ 100  REM  IT=NUMBER OF EQUATIONS (ROWS)
➤ 110  REM  JT=NUMBER OF VARIABLES (COLUMNS)
  120  INPUT "NUMBER OF EQUATIONS: ";IT
  130 JT = IT + 1
➤ 140  REM  READ COEFFICIENTS BY ROW
  150  FOR I = 1 TO IT
  155  PRINT
  156  PRINT "   EQUATION ";I
  157  PRINT
  160  FOR J = 1 TO JT
  190  IF J = JT GOTO 220
  200  PRINT "   COEFFICIENT ";J
  210  PRINT "   A(";I;",";J;"): "
➤ 211  INPUT A(I,J)
  215  GOTO 240
  220  PRINT "   CONSTANT ";J
  230  PRINT "   C(";I;"): "
  231  INPUT A(I,J)
  240  PRINT
  241  NEXT J
  250  NEXT I
  255  GOSUB 2000
  256  GOSUB 5000
➤ 260  REM  SET ROW INDEX
  264  PRINT
  265  PRINT
  266  PRINT "PROCESSING"
  268  PRINT
  269  PRINT
  270  FOR I = 1 TO IT
  280 L(I) = I
  285  NEXT I
➤ 290  REM  FIND PIVOT ROW FOR EACH COLUMN
  295  FOR J = 1 TO IT
  300  GOSUB 3000
➤ 310  REM  CHECK MAGNITUDE OF PIVOT
➤ 320  REM  ABORT IF PIVOT IS TOO SMALL
  330  IF  ABS (A(M,J)) < .0001 GOTO 360
  340 K(J) = M
  345  GOTO 380
  360  PRINT "PIVOT TOO SMALL - ABORT"
  370  GOTO 9998
➤ 380  REM  REDUCE PIVOT ROW STARTING AT COLUMN J+1
  390 JI = J + 1
  395 KP = A(M,J)
  400  FOR JP = JI TO JT
  410 A(M,JP) = A(M,JP) / KP
  420  NEXT JP
➤ 430  REM  REDUCE OTHER ROWS STARTING AT COLUMN J+1
  440  FOR IP = 1 TO IT
  450  IF IP = M GOTO 490
  460  FOR JP = JI TO JT
  465 KK = A(IP,J)
  470 A(IP,JP) = A(IP,JP) - A(M,JP) * KK
  480  NEXT JP
  490  NEXT IP
  500  NEXT J
➤ 510  REM  STORE VALUES
  520  FOR I = 1 TO IT
  530 M = K(I)
  540 X(I) = A(M,JT)
  550  NEXT I
➤ 560  REM  PRINT ANSWERS
  565  HOME
  570  PRINT
  580  PRINT "RESULTS"
  590  PRINT
  600  FOR I = 1 TO IT
  610  PRINT "X(";I;")=";X(I)
  620  NEXT I
  621  GOSUB 6000
  625  PRINT
  626  PRINT
➤ 630  INPUT "WANT TO CHECK RESULTS? (Y/N): ";CK$
  640  IF CK$ = "N" GOTO 9998
  650  IF CK$ = "Y" GOTO 680
  670  GOTO 630
  680  GOSUB 2500
  690  PRINT
  700  PRINT "ACCURACY CHECK"
  710  PRINT
  720  FOR I = 1 TO IT
  730  PRINT
  740  PRINT "     INPUT LOAD= ";A(I,JT)
```

MATRIX1

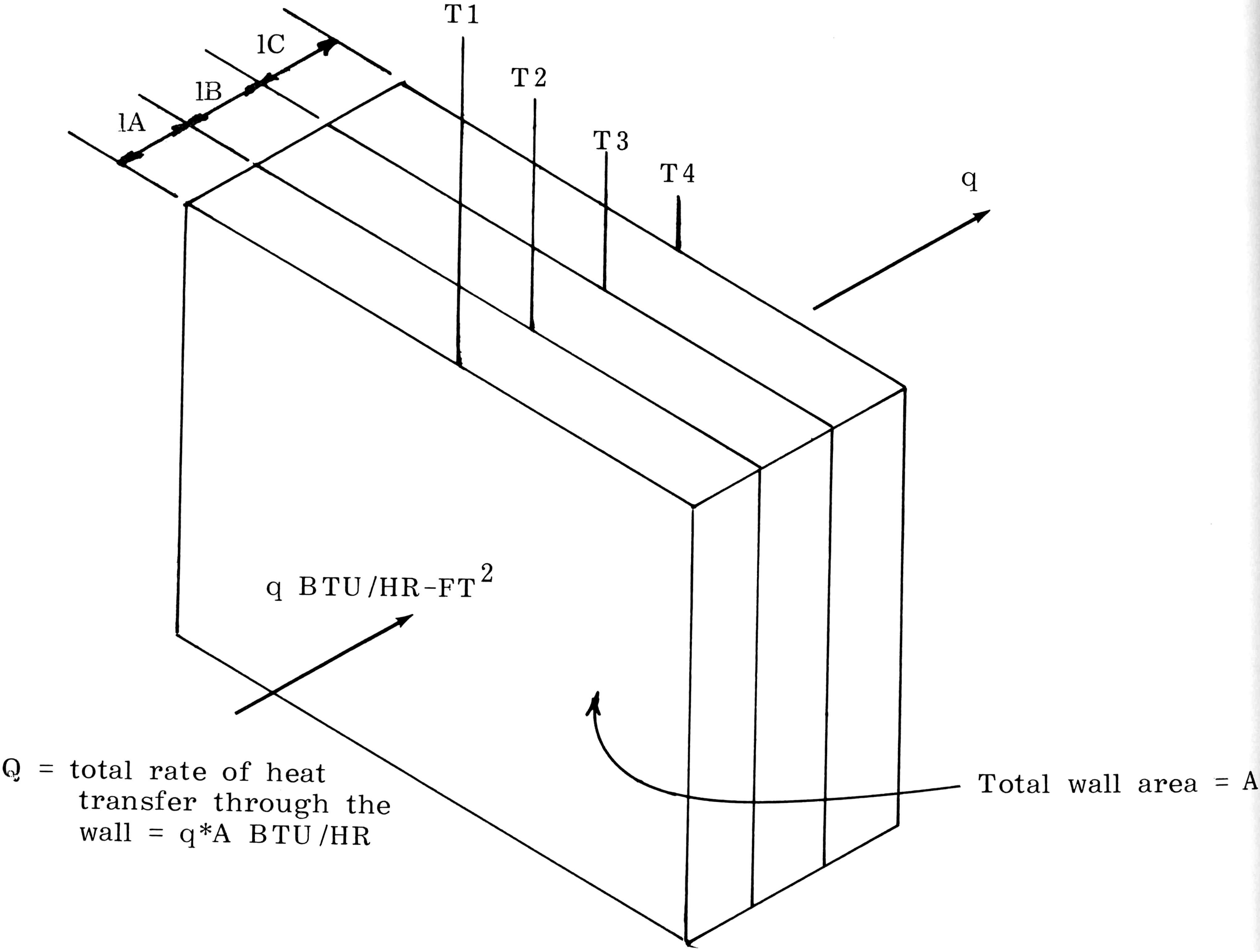
```
750  PRINT "CALCULATED LOAD= ";C(I)
755 D = C(I) - A(I,JT)
760  PRINT "     DIFFERENCE= ";D
770  NEXT I
780  GOTO 9998
2000  REM  SAVE DATA ON DISK FOR CHECKING LATER
2010  INPUT "WANT TO SAVE DATA ON DISK? (Y/N): ";CK$
2020  IF CK$ = "Y" THEN  GOTO 2060
2030  IF CK$ = "N" THEN  GOTO 2499
2040  PRINT "TRY AGAIN"
2050  GOTO 2010
2060  INPUT "FILE NAME: ";F$
2070  INPUT "DRIVE NUMBER: ";DR
2080 D$ = "": REM  CTRL-D
2090  PRINT D$;"OPEN";F$;",D";DR
2100  PRINT D$;"WRITE";F$
2110  PRINT IT
2120  PRINT JT
2130  FOR I = 1 TO IT
2140  FOR J = 1 TO JT
2150  PRINT A(I,J)
2160  NEXT J
2170  NEXT I
2180  PRINT D$;"CLOSE"
2499  RETURN
2500  REM  CALCULATE C(I),CHECK VALUES
2501  REM  RETRIEVE MATRIX FROM DISK
2510  INPUT "FILE NAME: ";F$
2511  INPUT "DRIVE NUMBER: ";DR
2515 D$ = "": REM  CTRL-D
2520  PRINT D$;"OPEN";F$;",D";DR
2530  PRINT D$;"READ";F$
2540  INPUT IT
2550  INPUT JT
2560  FOR I = 1 TO IT
2570  FOR J = 1 TO JT
2580  INPUT A(I,J)
2590  NEXT J
2600  NEXT I
2610  PRINT D$;"CLOSE"
2611  IF XX = 1 THEN  GOTO 2669
2615  REM  CALCULATE CHECK VALUES
2620  FOR I = 1 TO IT
2630  FOR J = 1 TO IT
2640 C(I) = C(I) + A(I,J) * X(J)
2650  NEXT J
2660  NEXT I
2669 XX = 0
2670  RETURN
3000  REM
3020  FOR I = 1 TO IT
3030  IF L(I) = 0 THEN  GOTO 3050
3035  REM  Y(I) IS ROW I
3040 Y(I) =  ABS (A(I,J))
```

MATRIX1

```
3045  GOTO 3055
3050 Y(I) = 0
3055  NEXT I
3060  REM  FIND MAX
3070 MM = 0
3080  FOR I = 1 TO IT
3090  IF Y(I) > MM THEN M = I
3095  IF Y(I) > MM THEN MM = Y(I)
3100  NEXT I
3110 L(M) = 0
3120  RETURN
5000  REM  PRINT MATRIX
5010  INPUT "WANT TO PRINT MATRIX? (Y/N): ";CK$
5011  IF CK$ = "N" THEN  GOTO 5110
5012  IF CK$ = "Y" THEN  GOTO 5020
5014  GOTO 5010
5020  PRINT D$;"PR#1"
5021  PRINT
5022  PRINT
5023  PRINT "INPUT MATRIX"
5024  PRINT
5030  FOR I = 1 TO IT
5040  PRINT
5050  PRINT "ROW ";I
5060  FOR J = 1 TO JT
5070  PRINT "      A(";I;",";J;")= ";A(I,J)
5080  NEXT J
5090  NEXT I
5100  PRINT D$;"PR#0"
5110  RETURN
6000  REM  PRINT X(I) ON PRINTER
6010  PRINT
6020  INPUT "WANT TO PRINT X(I)? (Y/N): ";CK$
6030  IF CK$ = "N" GOTO 6120
6040  IF CK$ = "Y" GOTO 6070
6060  GOTO 6020
6070  PRINT D$;"PR#1"
6072  PRINT
6073  PRINT
6074  PRINT "RESULTS"
6075  PRINT
6080  FOR I = 1 TO IT
6090  PRINT
6100  PRINT "    X(";I;")= ";X(I)
6105  NEXT I
6110  PRINT D$;"PR#0"
6120  RETURN
9998  PRINT "ALL DONE"
9999  END
```

THERMAL ANALYSIS

In this section we apply program *MATRIX1* to the solution of a common heat transfer problem. The model is shown in the sketch below:



Q = total rate of heat transfer through the wall = q*A BTU/HR

Heat flows through a wall having three layers. The inside temperature is T1; the outside temperature is T4. The temperatures at the interfaces between the layers are T2 and T3. We assume steady state heat transfer in which case the rate of heat flow through each layer is given by Fourier's Law of heat conduction:

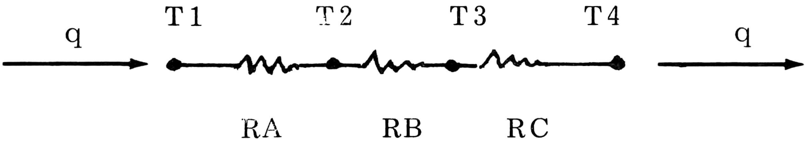
$$q = (k/l)*(TA - TB)$$

where q is the rate of heat transfer per unit area; k is the material's thermal conductivity and (TA-TB) is the temperature difference across the layer which has thickness l. This is analogous to the familiar Ohm's Law governing the flow of electric current through a resistor:

$$i = (1/R)*(EA - EB)$$

where i is the current (rate of flow of electric charge), R is the electrical resistance, and EA-EB is the voltage drop across the resistor.

Each layer in the wall may thus be represented as a thermal resistance:



We can write three equations for each resistor. Since this analysis assumes steady state conditions, the temperature and rate of heat flow through each layer is constant.

a) $q = (1/RA)*(T1 - T2)$

b) $q = (1/RB)*(T2 - T3)$

c) $q = (1/RC)*(T3 - T4)$

We will specify the inside and outside temperatures T1 and T4 and solve the above equations for q, the rate of heat flow through the wall, and the interlayer temperatures T2 and T3.

First, each equation must be put into the proper form for insertion into *MATRIX1*:

$$A(I,1)*T2 + A(I,2)*T3 + A(I,3)*q = A(I,4)$$

a) (1)*T2 + (0)*T3 + (RA)*q = (1)*T1

b) (1)*T2 + (-1)*T3 + (-RB)*q = 0

c) (0)*T2 + (1)*T3 + (-RC)*q = (1)*T4

We will solve these three equations using *MATRIX1*. When we run *MATRIX1,* we will enter values for the coefficients that are on the left side of the equations plus values for T1 and T4 on the right side, which we might regard as loadings. We'll use the following values:

T1 = 65^0 F
T2 = 30^0 F
RA = lA/kA = .25/.12 = 2.0833 (3" of wood)
RB = lB/kB = .0833/.024 = 3.4708 (1" of fiber board)
RC = lC/kC = .5/.15 = 3.3333 (6" of brick)

Units of q are btu/hr-ft^2. The total rate of heat loss through the wall is the rate of heat flow per unit area, q, multiplied by the wall area.

Using the above values, equations (a), (b) and (c) become:

a) (1)*T2 + (0)*T3 + (2.0833)*q = 65

b) (1)*T2 + (-1)*T3 + (-3.4708)*q = (0)

c) (0)*T2 + (1)*T3 + (-3.3333)*q = 30

SAMPLE RUN

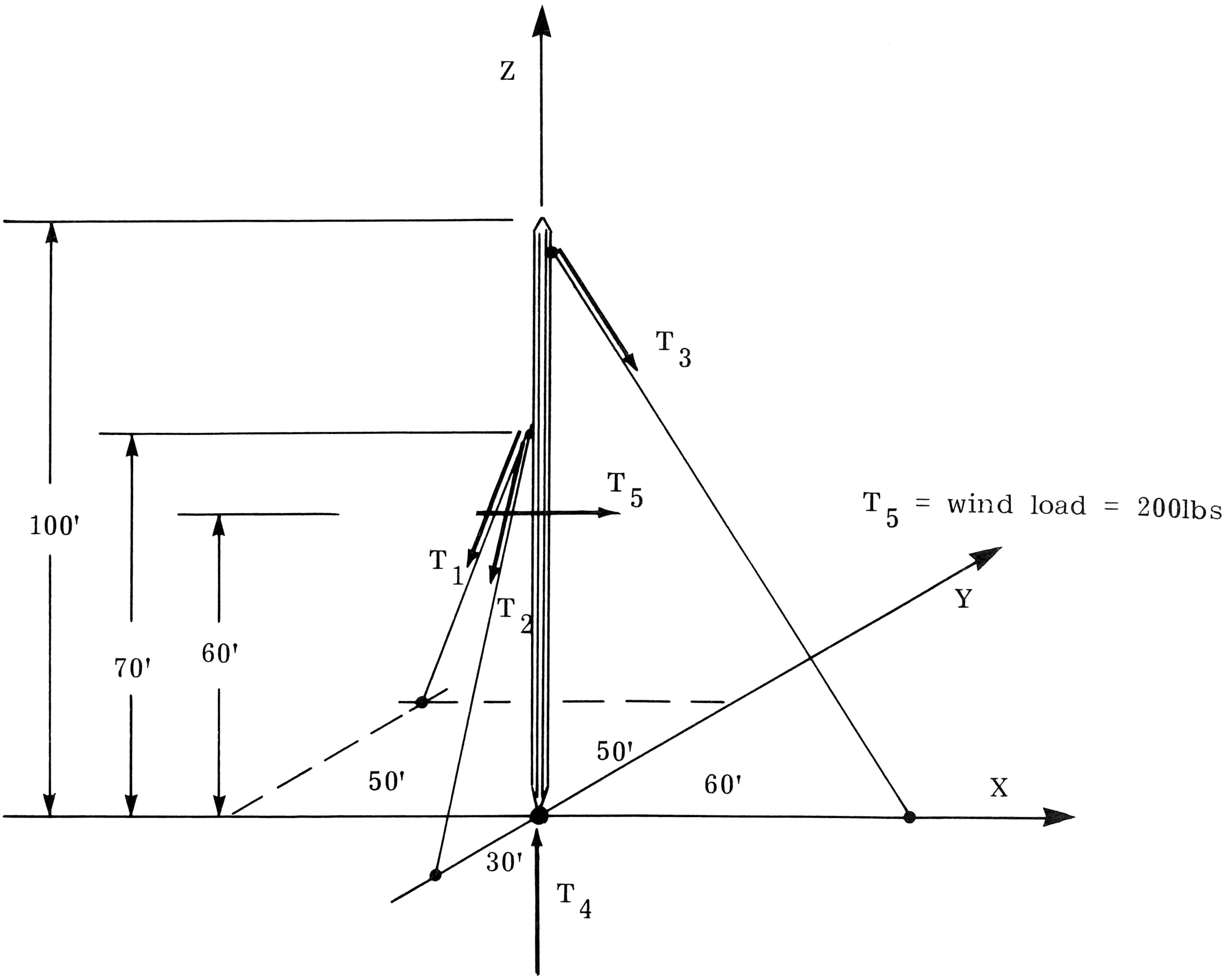
```
]RUN
MATRIX1
SOLVES A SET OF
LINEAR EQUATIONS BY
THE GAUSS-JORDAN TECHNIQUE
TYPE CONT TO CONTINUE

BREAK IN 55
]CONT
WANT TO READ MATRIX FROM DISK? (Y/N): N
NUMBER OF EQUATIONS: 3

   EQUATION 1

   COEFFICIENT 1
   A(1,1):
?1

   COEFFICIENT 2
   A(1,2):
?0

   COEFFICIENT 3
   A(1,3):
?.20833

   CONSTANT 4
   C(1):
?65


   EQUATION 2

   COEFFICIENT 1
   A(2,1):
?1

   COEFFICIENT 2
   A(2,2):
?-1

   COEFFICIENT 3
   A(2,3):
?-3.4708

   CONSTANT 4
   C(2):
?0
```

```
    EQUATION 3

    COEFFICIENT 1
    A(3,1):
?0

    COEFFICIENT 2
    A(3,2):
?1

    COEFFICIENT 3
    A(3,3):
?-3.3333

    CONSTANT 4
    C(3):
?30

WANT TO SAVE DATA ON DISK? (Y/N): Y
FILE NAME: MATRIX1THERMAL
DRIVE NUMBER: 1
WANT TO PRINT MATRIX? (Y/N): Y


INPUT MATRIX


ROW 1
        A(1,1)= 1
        A(1,2)= 0
        A(1,3)= .20833
        A(1,4)= 65

ROW 2
        A(2,1)= 1
        A(2,2)= -1
        A(2,3)= -3.4708
        A(2,4)= 0

ROW 3
        A(3,1)= 0
        A(3,2)= 1
        A(3,3)= -3.3333
        A(3,4)= 30


RESULTS

                                   T2
     X(1)= 63.9601964  ←
                                   T3
     X(2)= 46.6369575  ←
                                   q
     X(3)= 4.99113717  ←
```

STRUCTURAL ANALYSIS

STRUCTURAL ANALYSIS

In this section we illustrate how *MATRIX1* can be used to solve problems commonly encountered in structural analysis. The sketch below shows a tower supported by three cables and loaded by a wind load of 200 pounds. Tension in the cables is T1, T2 and T3. The base of the tower is supported by a ball which exerts a vertical force T4. We would like to determine T1, T2, T3 and T4.



We can derive four equations for the four unknown forces using the requirements for equiliblium. That is, the sum of the forces and moments acting on the tower must be zero. The method can be found in any text on statics and won't be repeated here. The equations that result are:

(-.5)*T1 + (0)*T2 + (.51)*T3 + (0)*T4 = -200

(.5)*T1 + (-.39)*T2 + (0)*T3 + (0)*T4 = 0

(-.7)*T1 + (-.92)*T2 + (-.86)*T3 + (1)*T4 = 0

(-35)*T1 + (0)*T2 + (51)*T3 + (0)*T4 = -12000

These can be input directly into *MATRIX1* for solution.

```
]RUN
MATRIX1
SOLVES A SET OF
LINEAR EQUATIONS BY
THE GAUSS-JORDAN TECHNIQUE
TYPE CONT TO CONTINUE

BREAK IN 55
]CONT
WANT TO READ MATRIX FROM DISK? (Y/N): N
NUMBER OF EQUATIONS: 4

   EQUATION 1

   COEFFICIENT 1
   A(1,1):
?-.5

   COEFFICIENT 2
   A(1,2):
?0

   COEFFICIENT 3
   A(1,3):
?.51

   COEFFICIENT 4
   A(1,4):
?0

   CONSTANT 5
   C(1):
?-200
```

```
    EQUATION 2

    COEFFICIENT 1
    A(2,1):
?.5

    COEFFICIENT 2
    A(2,2):
?-.39

    COEFFICIENT 3
    A(2,3):
?0

    COEFFICIENT 4
    A(2,4):
?0


    CONSTANT 5
    C(2):
?0


    EQUATION 3

    COEFFICIENT 1
    A(3,1):
?-.7

    COEFFICIENT 2
    A(3,2):
?-.92

    COEFFICIENT 3
    A(3,3):
?-.86

    COEFFICIENT 4
    A(3,4):
?1

    CONSTANT 5
    C(3):
?0


    EQUATION 4

    COEFFICIENT 1
    A(4,1):
?-35
```

```
   COEFFICIENT 2
   A(4,2):
?0

   COEFFICIENT 3
   A(4,3):
?51

   COEFFICIENT 4
   A(4,4):
?0

   CONSTANT 5
   C(4):
?-12000


WANT TO SAVE DATA ON DISK? (Y/N): Y
FILE NAME: MATRIX1TOWER
DRIVE NUMBER: 1
WANT TO PRINT MATRIX? (Y/N): Y


INPUT MATRIX


ROW 1
      A(1,1)= -.5
      A(1,2)= 0
      A(1,3)= .51
      A(1,4)= 0
      A(1,5)= -200

ROW 2
      A(2,1)= .5
      A(2,2)= -.39
      A(2,3)= 0
      A(2,4)= 0
      A(2,5)= 0

ROW 3
      A(3,1)= -.7
      A(3,2)= -.92
      A(3,3)= -.86
      A(3,4)= 1
      A(3,5)= 0

ROW 4
      A(4,1)= -35
      A(4,2)= 0
      A(4,3)= 51
      A(4,4)= 0
      A(4,5)= -12000
```

```
RESULTS

        X(1)=  533.333333   T1

        X(2)=  683.760684   T2

        X(3)=  130.718954   T3

        X(4)=  1114.81146   T4
```

PROGRAM *MATRIX2*

This short program can be used to update a matrix that has been previously stored on disk. This is a useful feature that can be used to change a system's matrix or its loadings without having to type in the entire matrix.

Most of the routines in this program have been developed earlier. A new module is subroutine 6000 which does the updating. It operates quite simply: It asks for the I,J position of the element that is to be changed. All you have to do is type in the I and J values and the new value of the element at that location.

The normal procedure is to first create the matrix using *MATRIX1* and store it on disk. If it is to be modified, load *MATRIX2*, recall the original matrix from disk, carry out the updating, and replace it on disk. *MATRIX1* can then be reloaded and a new solution obtained.

In the sample run that follows, the matrix that was used in the previous run, MATRIX1TOWER, is recalled and the loading is doubled. This means we double components A(1,5) and A(4,5). The updated matrix is then replaced on the disk as file MATRIX2TOWER.

```
5  HOME
10  PRINT "MATRIX2"
20  PRINT "RECALLS A MATRIX FROM DISK"
30  PRINT "UPDATES IT"
40  PRINT "REPLACES IT IN DISK"
50  PRINT "TYPE CONT TO CONTINUE"
55  STOP
56  DIM A(20,20)
57  REM
```

MATRIX2

```
58  REM
59  REM
60  REM
61  REM  INPUT
64  PRINT
65  PRINT "R=RECALL"; TAB( 12);"L=LIST"
75  PRINT "U=UPDATE"; TAB( 12);"F=FILE"
80  PRINT "E=END"
85  PRINT
90  INPUT "COMMAND: ";CK$
91  PRINT
92  IF CK$ = "E" THEN  GOTO 9998
95  IF CK$ = "R" THEN  GOSUB 2500
100  IF CK$ = "L" THEN  GOSUB 5000
105  IF CK$ = "U" THEN  GOSUB 6000
110  IF CK$ = "F" THEN  GOSUB 2000
120  GOTO 64
1996  REM
1997  REM
1998  REM
1999  REM
2000  REM  FILE MATRIX ON DISK
2001  PRINT "FILE"
2060  INPUT "FILE NAME: ";F$
2070  INPUT "DRIVE NUMBER: ";DR
2080 D$ = "": REM  CTRL-D
2090  PRINT D$;"OPEN";F$;",D";DR
2100  PRINT D$;"WRITE";F$
2110  PRINT IT
2120  PRINT JT
2130  FOR I = 1 TO IT
2140  FOR J = 1 TO JT
2150  PRINT A(I,J)
2160  NEXT J
2170  NEXT I
2180  PRINT D$;"CLOSE"
2190  RETURN
2496  REM
2497  REM
2498  REM
2499  REM
2500  REM  RECALL MATRIX FROM DISK
2501  PRINT "RECALL"
2510  INPUT "FILE NAME: ";F$
2511  INPUT "DRIVE NUMBER: ";DR
2515 D$ = "": REM  CTRL-D
2520  PRINT D$;"OPEN";F$;",D";DR
2530  PRINT D$;"READ";F$
2540  INPUT IT
2550  INPUT JT
➤ 2560  FOR I = 1 TO IT
➤ 2570  FOR J = 1 TO JT
➤ 2580  INPUT A(I,J)
```

```
2590  NEXT J
2600  NEXT I
2610  PRINT D$;"CLOSE"
2670  RETURN
4996  REM
4997  REM
4998  REM
4999  REM
5000  REM  LIST MATRIX
5001  PRINT "MATRIX ";F$
5020  PRINT D$;"PR#1"
5021  PRINT
5022  PRINT
5024  PRINT
5030  FOR I = 1 TO IT
5040  PRINT
5050  PRINT "ROW ";I
5060  FOR J = 1 TO JT
5070  PRINT "      A(";I;",";J;")= ";A(I,J)
5080  NEXT J
5090  NEXT I
5100  PRINT D$;"PR#0"
5110  RETURN
5996  REM
5997  REM
5998  REM
5999  REM
6000  REM  UPDATE
6001  PRINT "UPDATE"
6020  INPUT "   ROW I: ";I
6030  INPUT "COLUMN J: ";J
6040  INPUT "A(I,J): ";A(I,J)
6050  RETURN
9994  REM
9995  REM
9996  REM
9997  REM
9998  PRINT "ALL DONE"
9999  END
```

```
]RUN
MATRIX2
RECALLS A MATRIX FROM DISK
UPDATES IT
REPLACES IT IN DISK
TYPE CONT TO CONTINUE

BREAK IN 55
]CONT

R=RECALL    L=LIST
U=UPDATE    F=FILE
E=END
```

MATRIX2

```
COMMAND: R

RECALL
FILE NAME: MATRIX1TOWER
DRIVE NUMBER: 1

R=RECALL     L=LIST
U=UPDATE     F=FILE
E=END

COMMAND: L

MATRIX MATRIX1TOWER




ROW 1
      A(1,1)= -.5
      A(1,2)= 0
      A(1,3)= .51
      A(1,4)= 0
      A(1,5)= -200

ROW 2
      A(2,1)= .5
      A(2,2)= -.39
      A(2,3)= 0
      A(2,4)= 0
      A(2,5)= 0

ROW 3
      A(3,1)= -.7
      A(3,2)= -.92
      A(3,3)= -.86
      A(3,4)= 1
      A(3,5)= 0

ROW 4
      A(4,1)= -35
      A(4,2)= 0
      A(4,3)= 51
      A(4,4)= 0
      A(4,5)= -12000



ROW 1
      A(1,1)= -.5
      A(1,2)= 0
      A(1,3)= .51
      A(1,4)= 0
      A(1,5)= -400
```

```
ROW 2
          A(2,1)= .5
          A(2,2)= -.39
          A(2,3)= 0
          A(2,4)= 0
          A(2,5)= 0

ROW 3
          A(3,1)= -.7
          A(3,2)= -.92
          A(3,3)= -.86
          A(3,4)= 1
          A(3,5)= 0

ROW 4
          A(4,1)= -35
          A(4,2)= 0
          A(4,3)= 51
          A(4,4)= 0
          A(4,5)= -24000
```

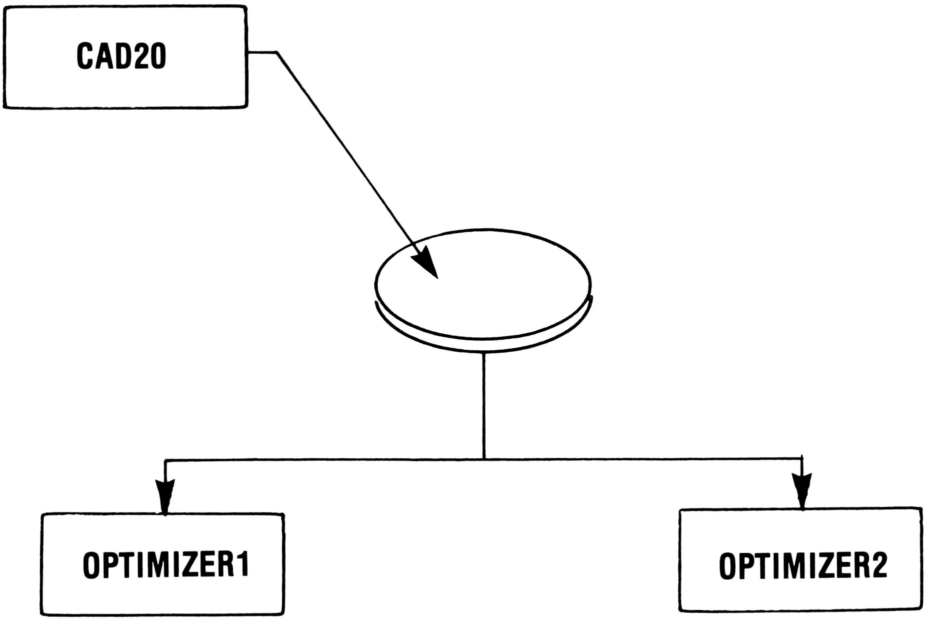
Reloading *MATRIX1* and re-running with these new values should double the solution values. This is verified by the following output from *MATRIX1*.

```
RESULTS


      X(1)= 1066.66667

      X(2)= 1367.52137

      X(3)= 261.437909

      X(4)= 2229.62293
```

Chapter 6: Optimization

Introduction, *6-1*

- **OPTIMIZER1:** Finds and plots the shortest path between a number of points using the "brute force" approach, *6-1*
- **OPTIMIZER2:** Finds and plots the shortest path between a number of points using the Monte Carlo method, *6-10*


CAD20
OPTIMIZER1
OPTIMIZER2

6-1

Optimization

OPTIMIZER1

INTRODUCTION

A typical optimization problem consists of a system that has a large number of possible states. The objective is to determine the optimum state; that is, the one that produces the optimum value of one or more parameters. Most often, the optimum will be a maximum or minimum value.

As an example, later in this chapter we will be analyzing a random walk problem. The system is a network of stations. A delivery man must stop at every station, returning to the one he started at. There are many different paths that can be taken. Each constitutes a different state. Since the parameter we wish to minimize is the distance traveled by the delivery man, we want the path that corresponds to the minimum distance.

In this chapter we present two methods for solving optimization problems such as this. The first is the "brute-force" approach where the software evaluates all possible states and selects the optimum. This approach is foolproof since we can program in any conditions and definitions of optimum and we are guaranteed the true optimum will be found. The main difficulty is that the computation time can be large, impractically large, if there are a great many states to be analyzed.

The second method is called Monte Carlo analysis. It is similar to the brute force approach except that we do not look at all possipossible states, and we look at say 30 percent of them, then there timum from that limited group. We are not guaranteed a true optimum, of course, but if the problem has a very large number of posible states, and we look at say 30 percent of them, then there is a good chance that we will get the optimum, or one very near to it.

PROGRAM *OPTIMIZER1*

This is an example of the brute force approach to optimization. While not the most elegant or efficient method, it has two distinct advantages: it will always find the optimum value and it is easy to program.

Program *OPTIMIZER1* has been specially written to find the shortest path between five points. This game is similar to the one that confronts a delivery man - starting at point 1, in what order should the five points be visited for the total distance traveled to be a minimum? This program solves the problem by trying all possible paths and selecting the one with the shortest distance. Conditions are that the delivery man always starts and ends at point 1.

A few words should be said about the brute force approach to solving problems of this type. Clearly, trying all possible solutions is not a very efficient way to solve this problem. There are other techniques that are more mathematically elegant and require less processing time. The problem with them is that the software is far more difficult and time consuming to write and debug. Most optimization problems require a specialized program. It is difficult to conceive of a generalized optimization program that would satisfy the majority of problems. So, even if a more advanced technique results in a few seconds or minutes less computer time, if it take several days longer to write and debug, it is hard to see where the gain in efficiency lies. The brute force approach, on the other hand, is easy to program and understand, can be adapted to a wide range of problems, requires little memory, runs in a reasonable amount of time, and always finds the correct answer.

PROGRAM LISTING

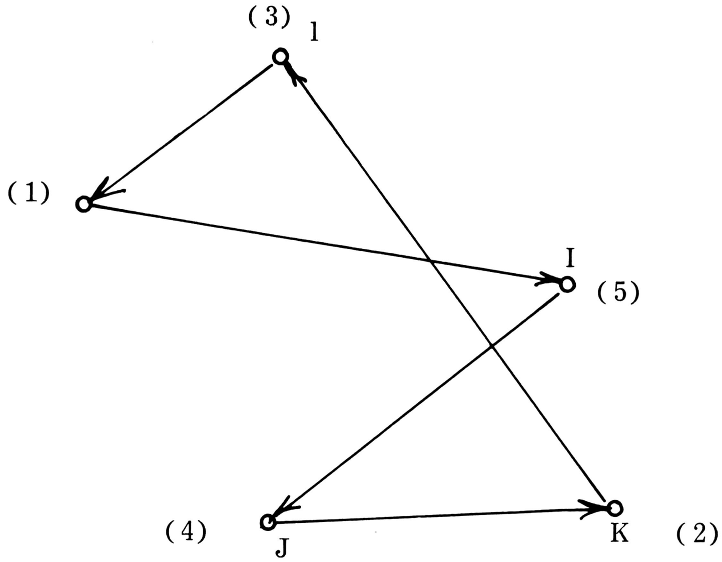
OPTIMIZER1 has been specialized to solve the random walk problem when there are five stations. The delivery man starts at station 1, visits all other stations, and returns to station 1. The question is, what sequence of stations, or what path, results in the shortest distance traveled?

The procedure is to first create the physical location of the points in real coordinates using CAD20. These are stored on disk. Then *OPTIMIZER1* is run. The drawing is read from disk, the stations are plotted on the screen, and then the shortest path is calculated and drawn.

Subroutine 3000 recalls the point drawing from disk. Note that we do not read lines in this routine but terminate the disk reading process after all points have been read. Subroutine 150 calculates the shortest path.

The parameter D in line 170 is set to a large number. As different paths are tried, the first will be compared with D. If it is smaller, then that new path will be saved. In this way, after all possible paths have been tried, the shortest is the one that will be saved.

There are four loops in the searching algorithm. They extend from line 180 to 360. Recall that the start point is point 1. Loop 180-360 selects the second point which may be points 2,3,4 or 5. Loop 190-350 selects the third point which also may be points 2,3,4 or 5. Loop 200-340 selects the fourth point and 207-335 the fifth. By nesting these loops, all possible combinations of points are generated. Starting the loops at point 2 insures that point 1 is not visited a second time. Line 195 makes sure that the third point is not visited a second time. Lines 205 and 206 make sure that the fourth point is not the same as either the second or the third. lines 208,209 and 210 insure that the fifth point is not the same as any of the previous.



Path for combination 1,5,4,2,3,1 - ie: I=5, j=4, K=2, L=3

For each combination of points, lines 212-305 calculate the total distance Q. Lines 212-230 calculate the distance from point 1 to the second point. Lines 240-260 calculate the distance from the second point to the third and add it to the distance from the first to the second. Similarly for the remainder of the algorithm.

Line 330 compares the total path length with the previous shortest path. If the new path is shorter, control drops to subroutine 1000 where the new path length and the point sequence are stored.

After all possible paths have been tried, the parameters ID, JD, KD and LD in lines 1010-1045 indicate the sequence of points producing the shortest path D. This path is then plotted in subroutine 2000.

In the sample run that follows the program listing, a mesh of five points was created with *CAD20* and stored on disk under the name CAD20POINTS. *OPTIMIZER1* was loaded and run. *CAD20POINTS* was recalled from disk and the optimization carried out.

Practically any logic can be incorporated in this program. For example, you might want to specify that point 4 always follows point 3. That would be an easy thing to include in the searching algorithm. Simply add a subroutine that checks each combination of points. It would find point 4 and check to see if it follows point 3. If not, the sequence is invalid and is discarded. The call to that subroutine would be right after line 210.

This program reads 3 dimensional data points but calculates only 2 dimensional paths. It would be a simple matter to extend it to 3 dimensional paths. Simply change the definition of Q in lines 212-305. For example, in lines 212-230 add a statement after 220 as follows:

```
225  C = Z(I) - Z(1)
```

and change 230:

```
230 = Q = SQR(A*A + B*B + C*C)
```

Similarly for the remainder of the loop.

```
10  PRINT "OPTIMIZER1"
20  PRINT "CALCULATES AND PLOTS"
30  PRINT "THE SHORTEST PATH BETWEEN"
40  PRINT "FIVE POINTS"
42  PRINT "USING THE BRUTE-FORCE"
43  PRINT "METHOD"
45  PRINT "TYPE CONT TO CONTINUE"
46  STOP
50  HGR
60  HCOLOR= 7
70  GOSUB 9990
80  DIM X(5),Y(5),Z(5)
85  DIM P(5),PC(5)
91  REM
```

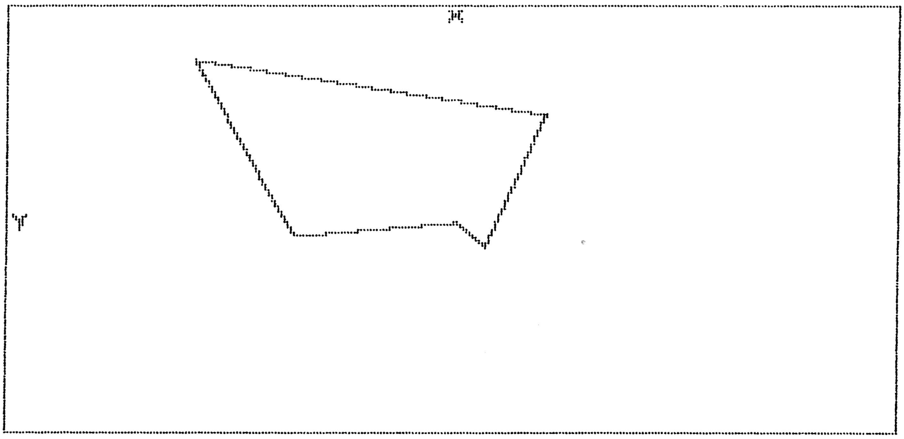
```
92  REM
93  REM
94  REM
95  REM  INPUT
96  PRINT "R=RECALL"; TAB( 12);"S=SEARCH"
97  PRINT "E=END"
98  INPUT "COMMAND: ";CK$
120  IF CK$ = "E" GOTO 9998
130  IF CK$ = "R" THEN  GOSUB 3000
140  IF CK$ = "S" THEN  GOSUB 150
142  GOTO 96
146  REM
147  REM
148  REM
149  REM
150  REM  CALCULATES THE SHORTEST PATH
160  REM  BY TRYING ALL POSSIBLE COMBINATIONS
161  PRINT "SEARCH"
➤ 170 D = 10000
180  FOR I = 2 TO 5
190  FOR J = 2 TO 5
195  IF J = I GOTO 350
200  FOR K = 2 TO 5
205  IF K = J GOTO 340
206  IF K = I GOTO 340
207  FOR L = 2 TO 5
208  IF L = K GOTO 335
209  IF L = J GOTO 335
210  IF L = I GOTO 335
212 A = X(I) - X(1)
220 B = Y(I) - Y(1)
230 Q =  SQR (A * A + B * B)
240 A = X(J) - X(I)
250 B = Y(J) - Y(I)
260 Q = Q +  SQR (A * A + B * B)
270 A = X(K) - X(J)
280 B = Y(K) - Y(J)
290 Q = Q +  SQR (A * A + B * B)
300 A = X(L) - X(K)
301 B = Y(L) - Y(K)
302 Q = Q +  SQR (A * A + B * B)
303 A = X(1) - X(L)
304 B = Y(1) - Y(L)
305 Q = Q +  SQR (A * A + B * B)
330  IF Q < D THEN  GOSUB 1000
335  NEXT L
340  NEXT K
350  NEXT J
360  NEXT I
370  GOSUB 2000
380  RETURN
996  REM
997  REM
```

```
998  REM
999  REM
1000  REM  STORE MINIMUM VALUES
1010 D = Q
1020 ID = I
1030 JD = J
1040 KD = K
1045 LD = L
1050  RETURN
1996  REM
1997  REM
1998  REM
1999  REM
2000  REM  PLOT PATH
2010 XR = X(1)
2020 YR = Y(1)
2030  GOSUB 5400
2040  HPLOT XT,YT
2050 XR = X(ID)
2060 YR = Y(ID)
2070  GOSUB 5400
2080  HPLOT  TO XT,YT
2090 XR = X(JD)
2100 YR = Y(JD)
2110  GOSUB 5400
2120  HPLOT  TO XT,YT
2130 XR = X(KD)
2140 YR = Y(KD)
2150  GOSUB 5400
2160  HPLOT  TO XT,YT
2170 XR = X(LD)
2180 YR = Y(LD)
2190  GOSUB 5400
2200  HPLOT  TO XT,YT
2210 XR = X(1)
2220 YR = Y(1)
2230  GOSUB 5400
2240  HPLOT  TO XT,YT
2250  PRINT "SHORTEST PATH= ";"1";ID;JD;KD;LD;"1"
2260  PRINT "PATH LENGTH= ";D
2270  RETURN
2996  REM
2997  REM
2998  REM
2999  REM
3000  REM  RECALL DRAWING FROM DISK
3010  INPUT "FILE NAME: ";F$
3020  INPUT "DRIVE NUMBER: ";DR
3030 D$ = "": REM  CTRL-D
3040  PRINT D$;"OPEN ";F$;",D";DR
3050  PRINT D$;"READ ";F$
3060  INPUT SF
3070  INPUT XO
3080  INPUT YO
```

```
3090  INPUT PX
3100  INPUT PY
3110  INPUT NP
3120  FOR I = 1 TO NP
3130  INPUT P(I)
3140  INPUT X(I)
3150  INPUT Y(I)
3160  INPUT Z(I)
3170  INPUT PC(I)
3180  NEXT I
3250  PRINT D$;"CLOSE"
3255  GOSUB 3500
3260  RETURN
3496  REM
3497  REM
3498  REM
3499  REM
3500  REM  PLOT
3510  FOR I = 1 TO NP
3520 XR = X(I)
3530 YR = Y(I)
3540  GOSUB 5400
3560  HCOLOR= PC(I)
3570  HPLOT XT,YT
3580  NEXT I
3740  RETURN
5396  REM
5397  REM
5398  REM
5399  REM
5400  REM  SCALE,ROTATE AND OFFSET PLOT COORDS
5410 XT = X0 + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = Y0 + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
9986  REM
9987  REM
9988  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

```
CAD20
CREATES A 3D DRAWING
FROM REAL PHYSICAL COORDS
SCALES, ROTATES AND OFFSETS
PLOT COORDINATES
TYPE CONT TO CONTINUE

BREAK IN 40
]CONT
SF: 1
XO,YO (APPLE UNITS): 140,80
PX: 0
PY: 180
P,L,FILE OR END: P
POINT NO: 1
X,Y,Z COORDS: 0,0,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 2
X,Y,Z COORDS: 30,40,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 3
X,Y,Z COORDS: 10,-10,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 5
X,Y,Z COORDS: -50,-5,0
COLOR: 7
P,L,FILE OR END: P
POINT NO: 4
X,Y,Z COORDS: -80,60,0
COLOR: 7
P,L,FILE OR END: FILE
FILE NAME: CAD20POINTS
DRIVE NUMBER: 1
P,L,FILE OR END: END
ALL DONE

]LOAD OPTIMIZER1
]RUN
OPTIMIZER1
CALCULATES AND PLOTS
THE SHORTEST PATH BETWEEN
FIVE POINTS
USING THE BRUTE-FORCE
METHOD
TYPE CONT TO CONTINUE

BREAK IN 46
]CONT
R=RECALL    S=SEARCH
E=END
```

```
COMMAND: R
FILE NAME: CAD2OPOINTS
DRIVE NUMBER: 1
R=RECALL    S=SEARCH
E=END
COMMAND: S
SEARCH
```



```
SHORTEST PATH= 132541
PATH LENGTH= 301.635666
```

PROGRAM *OPTIMIZER2*

OPTIMIZER2

Like *OPTIMIZER1,* this program also finds the shortest path between five points starting and ending at point 1 and not visiting the same point more than once. However, whereas *OPTIMIZER1* found the shortest path by calculating the path length for all possible paths (the brute force approach), *OPTIMIZER2* accomplishes the same result by selecting paths at random and choosing the one with the shortest length from that sample.

With the brute force approach, there are 24 possible paths. With the random selection method, the number of paths to be sampled is specified by the user. It is important to note that even if the number of paths selected is *greater* than 24, there is still no guarantee that the shortest will by found since some of the paths could be examined twice. An algorithm could be included in this program to make sure the same path is not examined twice, in which case trying 24 paths would insure that the optimum is found.

The random selection method employed in this program is termed the Monte Carlo approach. The name Monte Carlo derives from gambling terminology. The general idea is to pick a path at random and calculate the path length. Then select another path at random and calculate its path length. The process is somewhat like rolling dice to select the state of the system to be examined. The shortest path is then chosen from those examined. Instead of dice, the program uses a random number generator.

The algorithm works as follows: With the delivery man starting at point 1, the random number generator selects the next point to be visited. With the delivery man at the second point, the random number generator selects the third point to be visited, and so on. At each step, the next point chosen by the random number generator is examined to see if it has been visited before. If so, the generator is exercised again until a new acceptable point is found.

As peviously mentioned, there is no guarantee that the absolutely shortest path will be found with this method since it may not be among those examined. Of course, the greater the number of paths examined, the greater the probability the absolutely shortest will be found.

Why use the Monte Carlo method if there is no guarantee the optimum will be found? Because if the problem being examined has a great many states, this may be the only feasible technique. For example, in the problem of finding the shortest path between a number of points, it is not too difficult a job to write a program such as *OPTIMIZER1* that will examine all possible paths and give an exact answer. The problem is, the computer time required may be excessive if the number of points in the model is large. The Monte Carlo method, on the other hand, can be more easily extrapolated to larger problems and the running time can be kept within reasonable limits by choosing a modest number of paths to be examined. In most cases where there are a great many possible states in the model, the Monte Carlo method will usually give an answer close enough to the optimum for practicality.

PROGRAM LISTING

Program *OPTIMIZER2* is similar in structure to *OPTIMIZER1*. The main difference is that the loops where the point combinations were selected have been removed and replaced by one loop extending from line 175 to 360. This selects NT paths at random and calculates their length D. NT is the number of paths to be examined and is input in line 174.

Lines 185-207 use random numbers to choose the points comprising the paths. Point 1 is always the starting point. The second point is selected at random as follows: Line 185 calls subroutine 500 which generates a random digit. Line 510 uses the RND(x) command which returns a random number between 0 and 1 when x is positive. This command is available on the Apple computer and is a feature of most versions of BASIC, although the specific way it is used may vary from one system to another. Since we need a random digit greater than 1, we multiply the random number by 10 in line 520 and convert it to an integer in line 530.

Control then returns to line 186 where the number of the second point is set equal to this random digit. Lines 187 and 188 check to see if it lies between 2 and 5. Lines 190-207 carry out the same function for the remaining stops in the path.

The structure of this algorithm illustrates why it is sometimes called a "random walk". Each successive point in the path is selected at random by the random number generator. This is analogous to the random path of a drunk as he staggers from street corner to street corner.

As with *OPTIMIZER1,* we could easily build into this program any restrictions on the path we want. We could also extrapolate it to handle 3 dimensional paths with little difficulty.

An interesting exercise would be to modify this program to calculate the shortest path between an array of 3 dimensional points and store the points and lines for that path on disk, as we have done many times before. Then, load program *ISOVIEW* and see the path in 3 dimensions.

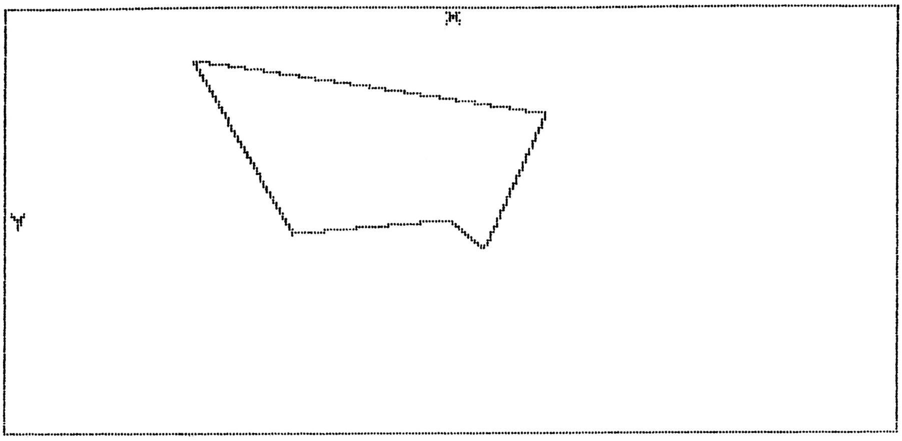
OPTIMIZER2

```
  10  PRINT "OPTIMIZER2"
  20  PRINT "CALCULATES AND PLOTS"
  30  PRINT "THE SHORTEST PATH BETWEEN"
  40  PRINT "FIVE POINTS"
  42  PRINT "USING THE MONTE-CARLO"
  43  PRINT "METHOD"
  45  PRINT "TYPE CONT TO CONTINUE"
  46  STOP
  50  HGR
  60  HCOLOR= 7
  70  GOSUB 9990
  80  DIM X(5),Y(5),Z(5)
  85  DIM P(5),PC(5)
  91  REM
  92  REM
  93  REM
  94  REM
  95  REM  INPUT
  96  PRINT "R=RECALL"; TAB( 12);"S=SEARCH"
  97  PRINT "E=END"
  98  INPUT "COMMAND: ";CK$
 120  IF CK$ = "E" GOTO 9998
 130  IF CK$ = "R" THEN  GOSUB 3000
 140  IF CK$ = "S" THEN  GOSUB 150
 142  GOTO 96
 146  REM
 147  REM
 148  REM
 149  REM
 150  REM  CALCULATES THE SHORTEST PATH
 160  REM  USING THE MONTE-CARLO TECHNIQUE
 161  PRINT "SEARCH"
 162  HGR
 163  GOSUB 9990
 170 D = 10000
► 174  INPUT "NUMBER OF TRYS: ";NT
 175  FOR II = 1 TO NT
 185  GOSUB 500
 186 I = A
 187  IF I < 2 GOTO 185
 188  IF I > 5 GOTO 185
 190  GOSUB 500
 191 J = A
 192  IF J < 2 GOTO 190
 193  IF J > 5 GOTO 190
 194  IF J = I GOTO 190
 195  GOSUB 500
 196 K = A
 197  IF K < 2 GOTO 195
 198  IF K > 5 GOTO 195
 199  IF K = J GOTO 195
 200  IF K = I GOTO 195
 201  GOSUB 500
 202 L = A
 203  IF L < 2 GOTO 201
 204  IF L > 5 GOTO 201
 205  IF L = K GOTO 201
 206  IF L = J GOTO 201
 207  IF L = I GOTO 201
 212 A = X(I) - X(1)
 220 B = Y(I) - Y(1)
 230 Q =  SQR (A * A + B * B)
 240 A = X(J) - X(I)
 250 B = Y(J) - Y(I)
 260 Q = Q +  SQR (A * A + B * B)
 270 A = X(K) - X(J)
 280 B = Y(K) - Y(J)
 290 Q = Q +  SQR (A * A + B * B)
 300 A = X(L) - X(K)
 301 B = Y(L) - Y(K)
 302 Q = Q +  SQR (A * A + B * B)
 303 A = X(1) - X(L)
 304 B = Y(1) - Y(L)
 305 Q = Q +  SQR (A * A + B * B)
 310  IF L = J GOTO 260
 320  IF L = I GOTO 260
 330  IF Q < D THEN  GOSUB 1000
 360  NEXT II
 370  GOSUB 2000
 380  RETURN
 496  REM
 497  REM
 498  REM
 499  REM
 500  REM  GENERATES A RANDOM DIGIT
 510 A =  RND (1)
 520 A = A * 10
 530 A =  INT (A)
 540  RETURN
 996  REM
 997  REM
 998  REM
 999  REM
 1000  REM  STORE MINIMUM VALUES
 1010 D = Q
 1020 ID = I
 1030 JD = J
 1040 KD = K
 1045 LD = L
 1050  RETURN
 1996  REM
 1997  REM
 1998  REM
 1999  REM
 2000  REM  PLOT PATH
 2010 XR = X(1)
 2020 YR = Y(1)
 2030  GOSUB 5400
```

```
2040  HPLOT XT,YT
2050 XR = X(ID)
2060 YR = Y(ID)
2070  GOSUB 5400
2080  HPLOT  TO XT,YT
2090 XR = X(JD)
2100 YR = Y(JD)
2110  GOSUB 5400
2120  HPLOT  TO XT,YT
2130 XR = X(KD)
2140 YR = Y(KD)
2150  GOSUB 5400
2160  HPLOT  TO XT,YT
2170 XR = X(LD)
2180 YR = Y(LD)
2190  GOSUB 5400
2200  HPLOT  TO XT,YT
2210 XR = X(1)
2220 YR = Y(1)
2230  GOSUB 5400
2240  HPLOT  TO XT,YT
2250  PRINT "SHORTEST PATH= ";"1";ID;JD;KD;LD;"1"; TAB( 25);"D= ";D
2270  RETURN
2996  REM
2997  REM
2998  REM
2999  REM
3000  REM  RECALL DRAWING FROM DISK
3010  INPUT "FILE NAME: ";F$
3020  INPUT "DRIVE NUMBER: ";DR
3030 D$ = "": REM  CTRL-D
3040  PRINT D$;"OPEN ";F$;",D";DR
3050  PRINT D$;"READ ";F$
3060  INPUT SF
3070  INPUT XO
3080  INPUT YO
3090  INPUT PX
3100  INPUT PY
3110  INPUT NP
3120  FOR I = 1 TO NP
3130  INPUT P(I)
3140  INPUT X(I)
3150  INPUT Y(I)
3160  INPUT Z(I)
3170  INPUT PC(I)
3180  NEXT I
3250  PRINT D$;"CLOSE"
3255  GOSUB 3500
3260  RETURN
3496  REM
3497  REM
3498  REM
3499  REM
3500  REM  PLOT
```

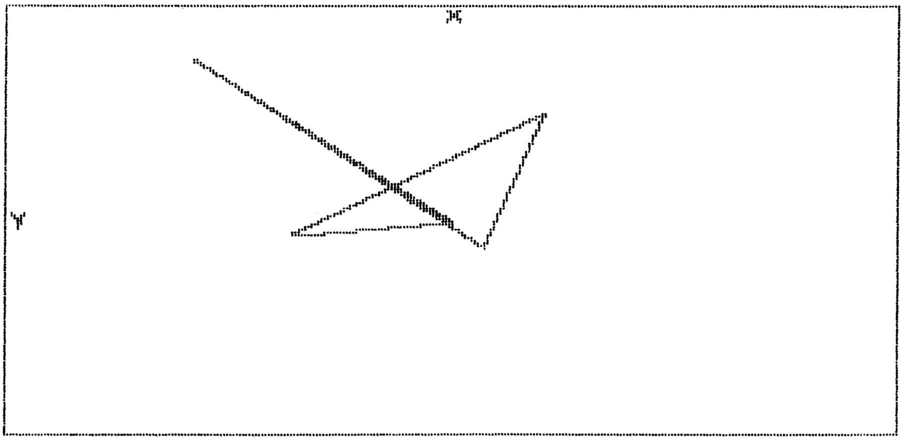
```
3510  FOR I = 1 TO NP
3520 XR = X(I)
3530 YR = Y(I)
3540  GOSUB 5400
3560  HCOLOR= PC(I)
3570  HPLOT XT,YT
3580  NEXT I
3740  RETURN
5396  REM
5397  REM
5398  REM
5399  REM
5400  REM  SCALE,ROTATE AND OFFSET PLOT COORDS
5410 XT = XO + SF * (XR *  COS (PX) - YR *  SIN (PY))
5420 YT = YO + SF * (XR *  SIN (PX) + YR *  COS (PY))
5430  RETURN
9986  REM
9987  REM
9988  REM
9989  REM
9990  HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
9991  HPLOT 138,2 TO 142,6
9992  HPLOT 142,2 TO 138,6
9993  HPLOT 2,77 TO 4,80 TO 4,83 TO 6,77
9994  RETURN
9998  PRINT "ALL DONE"
9999  END
```

```
]RUN
OPTIMIZER2
CALCULATES AND PLOTS
THE SHORTEST PATH BETWEEN
FIVE POINTS
USING THE MONTE-CARLO
METHOD
TYPE CONT TO CONTINUE

BREAK IN 46
]CONT
R=RECALL    S=SEARCH
E=END
COMMAND: R
FILE NAME: CAD2OPOINTS
DRIVE NUMBER: 1


R=RECALL    S=SEARCH
E=END
COMMAND: S
SEARCH
```



```
NUMBER OF TRYS: 24
SHORTEST PATH= 145231    D= 301.635666
```

```
R=RECALL    S=SEARCH
E=END
COMMAND: S
SEARCH
```



```
NUMBER OF TRYS: 4
SHORTEST PATH= 153241    D= 409.906368
```

INDEX

applications programs, *2-5*
binary data, *2-5*
CAD/CAM, *2-1*
CAD/CAM systems, advantages of, *2-10*
  CAD10, 2-11
  CAD20, 2-19
  CAD30, 2-30
  CAD50, 2-48
  CENTROID, 2-58
circles, *1-14*
coordinates, real physical, *2-20*
coordinates, screen, *2-11*
coordinate systems, *1-2*
coordinate transformations, *2-22*
crank angle, *3-2*
DECpc, *2-2*
drawing data base, *2-5*
drawing revisions, *2-9*
electric circuits, *5-1*
finite elements, *2-5*
follower angle, *3-2*
follower bar, *3-9*
four bar linkage, *3-1*
Fourier Analysis, *4-1*
Fourier Series, *4-1*
Fourier Transforms, *4-8*
  FOURIER1, 4-1
  FOURIER2, 4-8
Gauss-Jordan analysis, *5-1*
graphics modes, *1-4*
HCOLOR, *1-4*
heat transfer, *5-15*
HGR, *1-4*
histogram, *4-10*
HPLOT x1,y1, *1-5*
HPLOT x1,y1 TO x2,y2, *1-5*
HPLOT x1,y1 TO x2,y2 TO ..., *1-5*
HP85,87, *2-2*
IBMpc, *2-2*
interface program, *2-8*
  ISOVIEW, 2-65
limitations of micros, *2-3*
  LINKAGE1, 3-3
  LINKAGE2, 3-9
  LINKAGE3, 3-16
  LINKAGE4, 3-19
matrix methods, *5-1*
  MATRIX1, 5-9
  MATRIX2, 5-24
mechanisms, *3-1*
Monte Carlo, 2-3, *6-10*
optimization, *6-1*
  OPTIMIZER1, 6-1
  OPTIMIZER2, 6-10
  PLOT10, 1-5
  PLOT11, 1-6
  PLOT12, 1-6
  PLOT13, 1-7
  PLOT14, 1-7
  PLOT15, 1-8
  PLOT16, 1-9
  PLOT17, 1-10
  PLOT60, 1-14
  PLOT61, 1-16
  PLOT62, 1-18
pointer array, *2-13*
radians, *1-15*
random numbers, *6-10*
random walk, *6-11*
robots, *2-8*
sequential text file, *2-13*
simultaneous equations, *5-1*
slider crank, *3-19*
structural analysis, *5-20*
thermal analysis, *5-15*
TRS-80, *2-2*
vectors, *1-11*
Wang, *2-2*

ISBN 0-940254-05-0

ENGINEERING SOFTWARE FOR MICROS

B. J. Korites

KERN